Centrality Measures on Big Graphs: Exact, Approximated, and Distributed Algorithms

Francesco Bonchi[1,2]
Gianmarco De Francisci Morales[3]
Matteo Riondato[4]

[1]ISI Foundation, Turin (Italy)

[2]Eurecat, Technological Center of Catalonia, Barcelona (Spain)

[3]Qatar Computing Research Institute, Doha (Qatar)

[4]Two Sigma Investments LP, NYC (USA)

WWW'16 – Montréal, April 11–15, 2016

Slides available at

http://matteo.rionda.to/centrtutorial/

Acknowledgements

- Paolo Boldi
- Andreas Kaltenbrunner
- Evgenios M. Kornaropoulos
- Nicolas Kourtellis
- Eli Upfal
- Sebastiano Vigna

Roadmap

- Introduction
  - motivation, history, and definitions
  - closeness and betweenness centrality
  - axioms: what to look for in a centrality measure
- Exact algorithms
  - exact algorithms on static graphs
  - exact algorithms on dynamic graphs
- Approximation algorithms
  - approximation algorithms on static graphs
  - approximation algorithms on dynamic graphs
- Conclusions
  - open problems and research directions

Introduction

Social network analysis

- Social network analysis is the study of social entities and their interactions and relationships
- The interactions and relationships can be represented with a network or graph,
  - each vertex represents an actor
  - each link represents a relationship
- From the graph, we can study the properties of its structure, and the role, position, and prestige of each social entity.
- We can also find various kinds of sub-graphs, e.g., communities formed by groups of actors.

Centrality in networks

- Important or prominent actors are those that are extensively linked or involved with other actors
- A person with extensive contacts (links) or communications with many other people in the organization is considered more important than a person with relatively fewer contacts
- A central actor is one involved in many ties
- Graph centrality is a topic of uttermost importance in social sciences
- Also related to the problem of ranking in the context of Web Search:
  - Each webpage is a social actor
  - Each hyperlink is an endorsement relationship
  - Centrality measures provide a query independent link-based score of importance of a web page

History of centrality (in a nutshell)

- first attempts in the late 1940s at MIT (Bavelas 1946), in the framework of communication patterns and group collaboration;
- in the following decades, various measures of centralities were proposed and employed by social scientists in a myriad of contexts (Bavelas 1951; Katz 1953; Shaw 1954; Beauchamp 1965; Mackenzie 1966; Burgess 1969; Anthonisse 1971; Czapiel 1974...) item a new interest raised in the mid-90s with the advent of search engines: a "reincarnation" of centrality.

Freeman, 1979

"several measures are often only vaguely related to the intuitive ideas they purport to index, and many are so complex that it is difficult or impossible to discover what, if anything, they are measuring."

Types of centralities

Starting point: the central vertex of a star is the most important! Why?

1. the vertex with largest degree;
2. the vertex that is closest to the other vertexes (e.g., that has the smallest average distance to other vertexes);
3. the vertex through which all shortest paths pass;
4. the vertex with the largest number of incoming paths of length k, for every k;
5. the vertex that maximizes the dominant eigenvector of the graph adjacency matrix;
6. the vertex with highest probability in the stationary distribution of the natural random walk on the graph.

These observations lead to corresponding competing views of centrality.

Types of centralities

This observation leads to the following classes of indices of centrality:

1. measures based on distances [degree, closeness, Lin's index];
2. measures based on paths [betweenness, Katz's index];
3. spectral measures [dominant eigenvector, Seeley's index, PageRank, HITS, SALSA].

The last two classes are largely the same (even if that wasn't fully understood for a long time.)

Geometric centralities

- degree (folklore): $c_{\mathrm{deg}}(x) = d^-(x)$
- closeness (Bavelas, 1950): $c_{\mathrm{clos}}(x) = \mathsf{c}(x) = \frac{1}{\sum_y d(y,x)}$
- Lin (Lin, 1976): $c_{\mathrm{Lin}}(x) = \frac{r(x)^2}{\sum_y d(y,x)}$ where $r(x)$ is the number of vertexes that are co-reachable from x
- harmonic (Boldi and Vigna, 2013) $c_{\mathrm{harm}}(x) = \sum_{y \neq x} \frac{1}{d(y,x)}$

Path-based centralities

- betweenness (Anthonisse, 1971): $c_{\mathrm{bet}}(x) = \mathsf{b}(x) = \sum_{y,z \neq x, \sigma_{yz} \neq 0} \frac{\sigma_{yz}(x)}{\sigma_{yz}}$ where σ_{yz} is the number of shortest paths $y \to z$, and $\sigma_{yz}(x)$ is the number of such paths passing through x
- Katz (Katz, 1951): $c_{\mathrm{Katz}}(x) = \sum_{t \geq 0} \beta^t p_t(x)$ where $p_t(x)$ is the number of paths of length t ending in x, and β is a parameter ($\beta < 1/\rho$)

Spectral centralities

- dominant (Wei, 1953): $c_{\mathrm{dom}}(x)$ is the dominant (right) eigenvector of G
- Seeley (Seeley, 1949): $c_{\mathrm{Seeley}}(x)$ is the dominant (left) eigenvector of G_r
- PageRank (Brin, Page et al., 1999): $c_{\mathrm{PR}}(x)$ is the dominant (left) eigenvector of $\alpha G_r + (1-\alpha)\mathbf{1}^T\mathbf{1}/n$ (where $\alpha < 1$)
- HITS (Kleinberg, 1997): $c_{\mathrm{HITS}}(x)$ is the dominant (left) eigenvector of $G^T G$
- SALSA (Lempel, Moran, 2001): $c_{\mathrm{SALSA}}(x)$ is the dominant (left) eigenvector of $G_c^T G_r$

Where G denotes the adjacency matrix of the graph, G_r is the adjacency matrix normalized by row, and G_c is the adjacency matrix normalized by column.

Closeness and Betweenness

Closeness centrality

Motivation

It measures the ability to quickly access or pass information through the graph;

Definition (Closeness Centrality)

- closeness centrality $c(x)$ of a vertex x

$$c(x) = \frac{1}{\sum_{y \neq x \in V} d(y,x).}$$

- $d(y,x)$ is the length of a shortest path between y and x.
- The closeness of a vertex is defined as the inverse of the sum of the Shortest Path (SP) distances between the vertex and all other vertexes of the graph.
- When multiplied by $n-1$, it is effectively the inverse of the average SP distance.

Betweenness centrality

Motivation

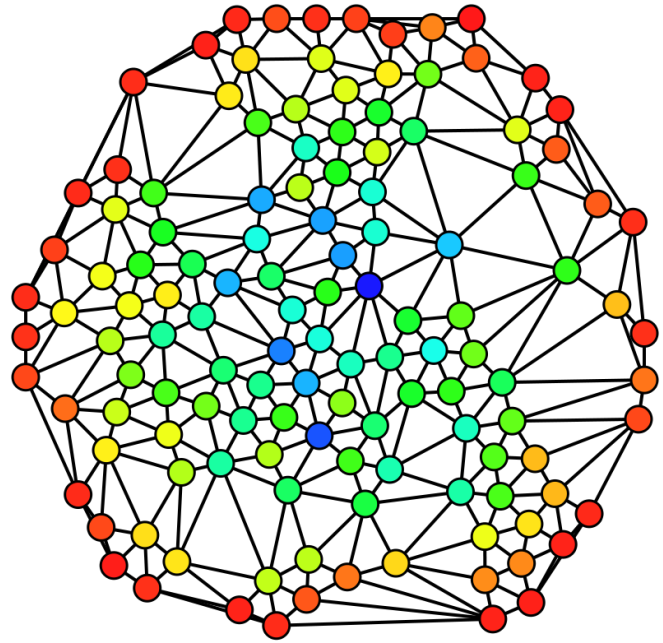
It measures the frequency with which a user appears in a shortest path between two other users.

Definition (Betweennes centrality)

- betweenness centrality $b(x)$ of a vertex x:

$$b(x) = \sum_{\substack{s \neq x \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(x)}{\sigma_{st}}$$

- σ_{st}: number of SPs from s to t
- $\sigma_{st}(x)$: how many of them pass through x



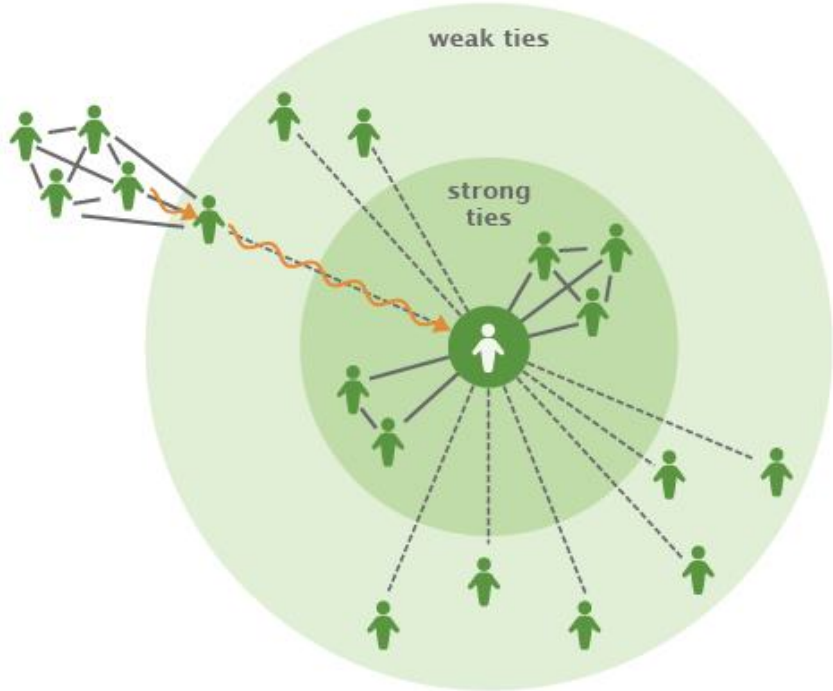
Example retrieved from Wikipedia

Betweenness centrality

- Can be defined also for edges (similarly to vertexes)
- Edges with high betweenness are known as "weak ties"
- They tend to act as bridges between two communities

The strength of weak ties (Granovetter 1973)

- Dissemination and coordination dynamics are influenced by links established to vertexes of different communities.
- The importance of these links has become more and more with the rise of social networks and professional networking platforms.



Weak ties

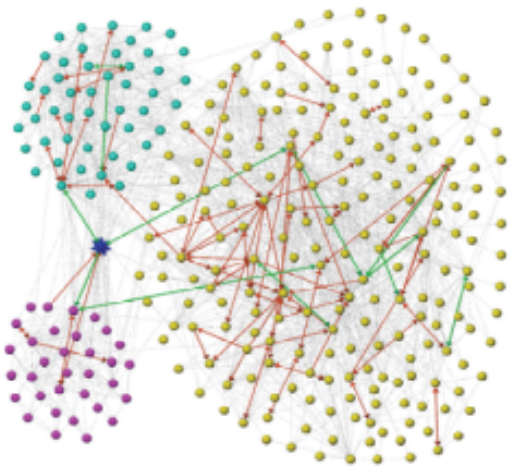
Bakshy et al. 2012

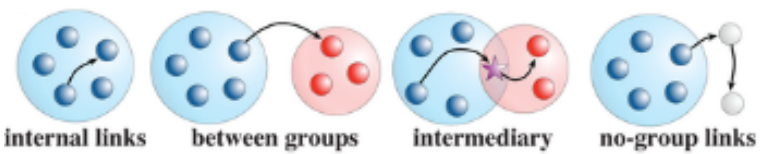
Weak links have a greater potential to expose links to new contacts that otherwise would not have been discovered.

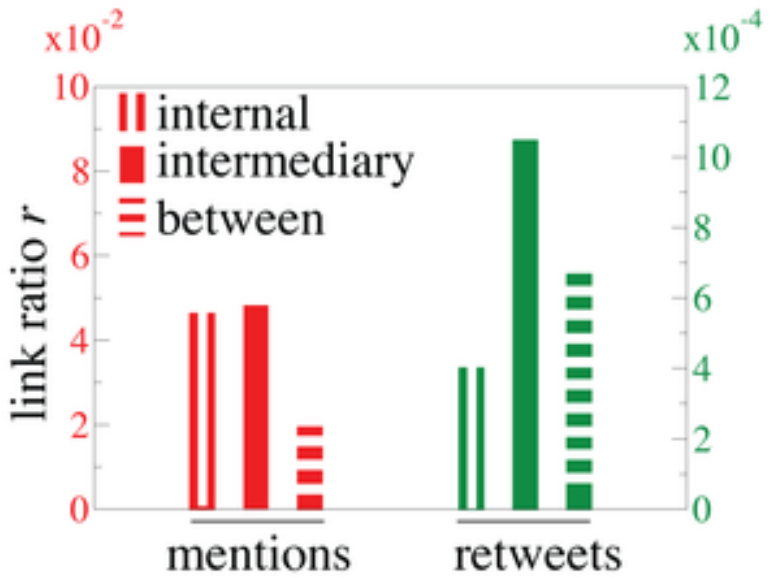
Weak ties

Grabowicz et al. 2012

- Personal interactions are more likely to occur in internal links within communities (strong links)
- Events or new information is propagated faster by intermediate links (weak links).



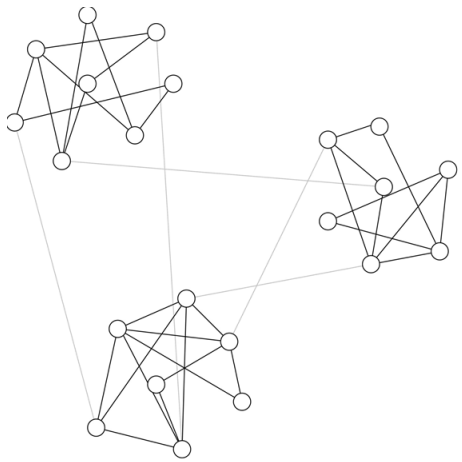


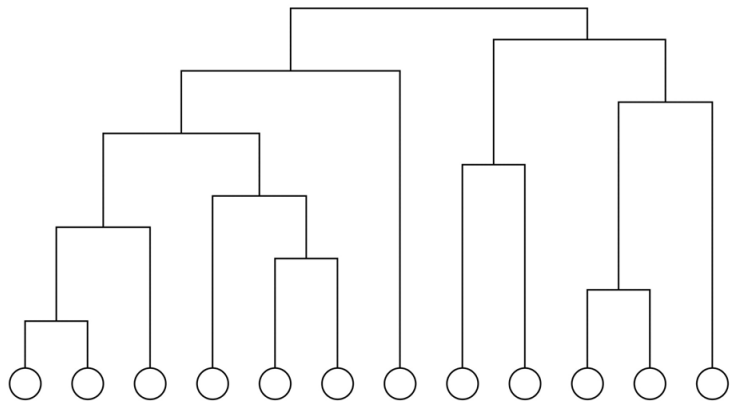


Girvan-Newman algorithm for community detection (Girvan and Newman 2002)

Hierarchical divisive clustering by recursively removing the "weakest tie":

1. Compute edge betweenness centrality of all edges;
2. Remove the edge with the highest betweenness centrality;
3. Repeat from 1.

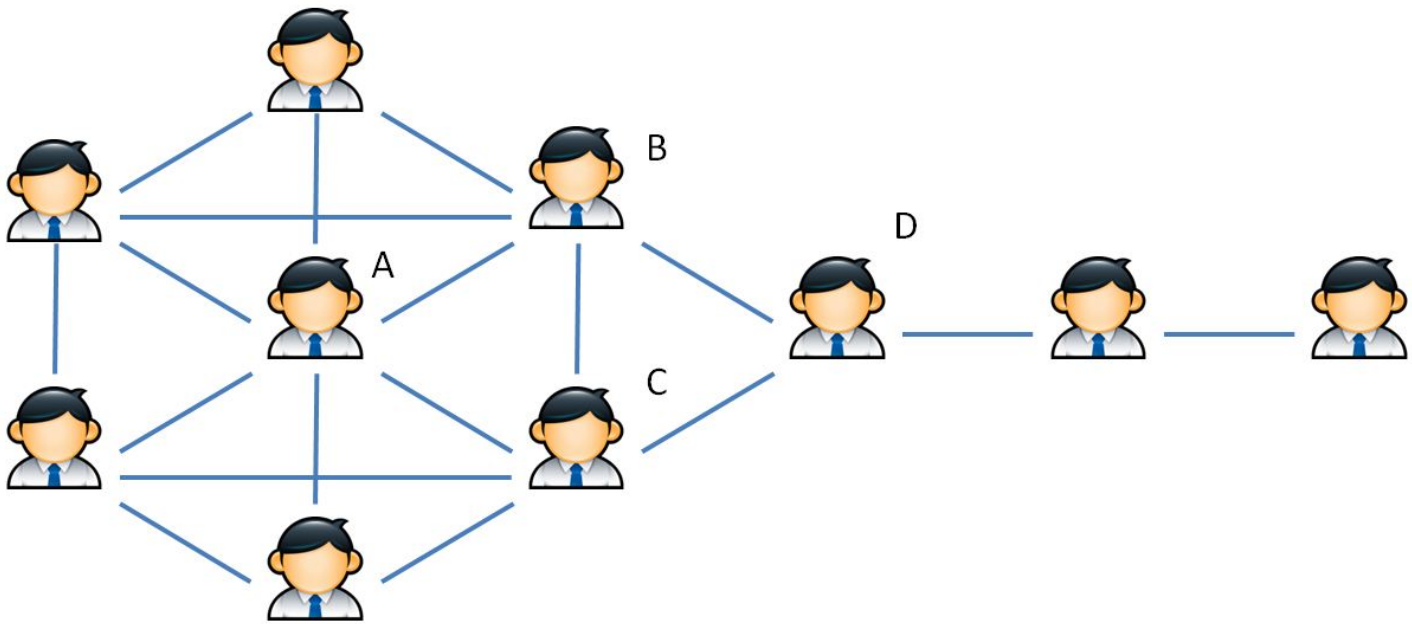




Comparison

Which vertex is the most central?

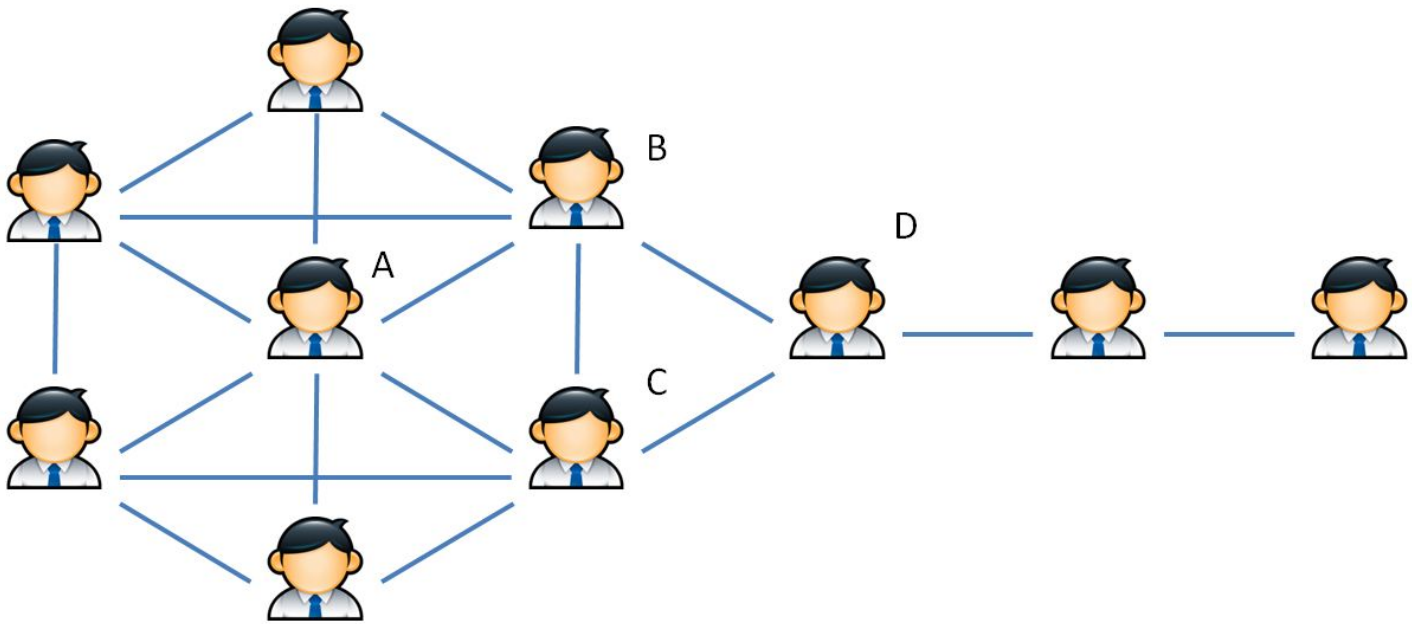
- for Degree Centrality:
- for Closeness Centrality:
- for Betweenness Centrality:



Comparison

Which vertex is the most central?

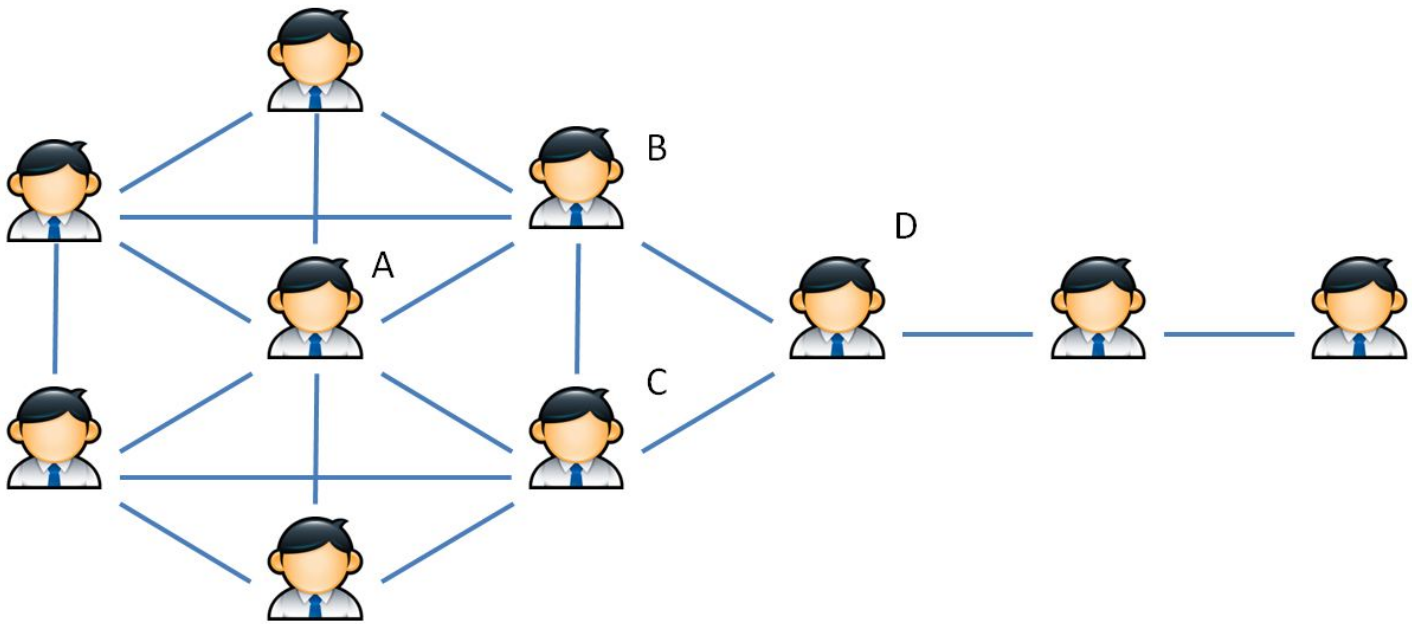
- for Degree Centrality: user A
- for Closeness Centrality:
- for Betweenness Centrality:



Comparison

Which vertex is the most central?

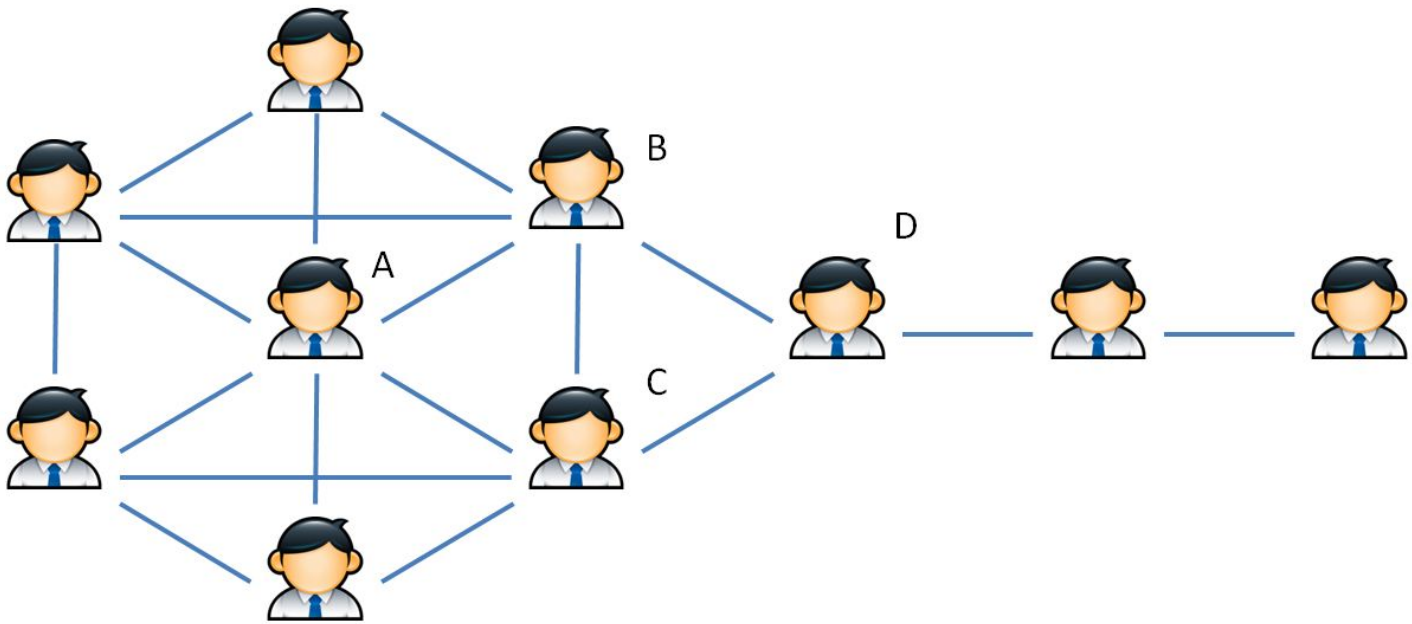
- for Degree Centrality: user A
- for Closeness Centrality: users B and C
- for Betweenness Centrality:



Comparison

Which vertex is the most central?

- for Degree Centrality: user A
- for Closeness Centrality: users B and C
- for Betweenness Centrality: user D

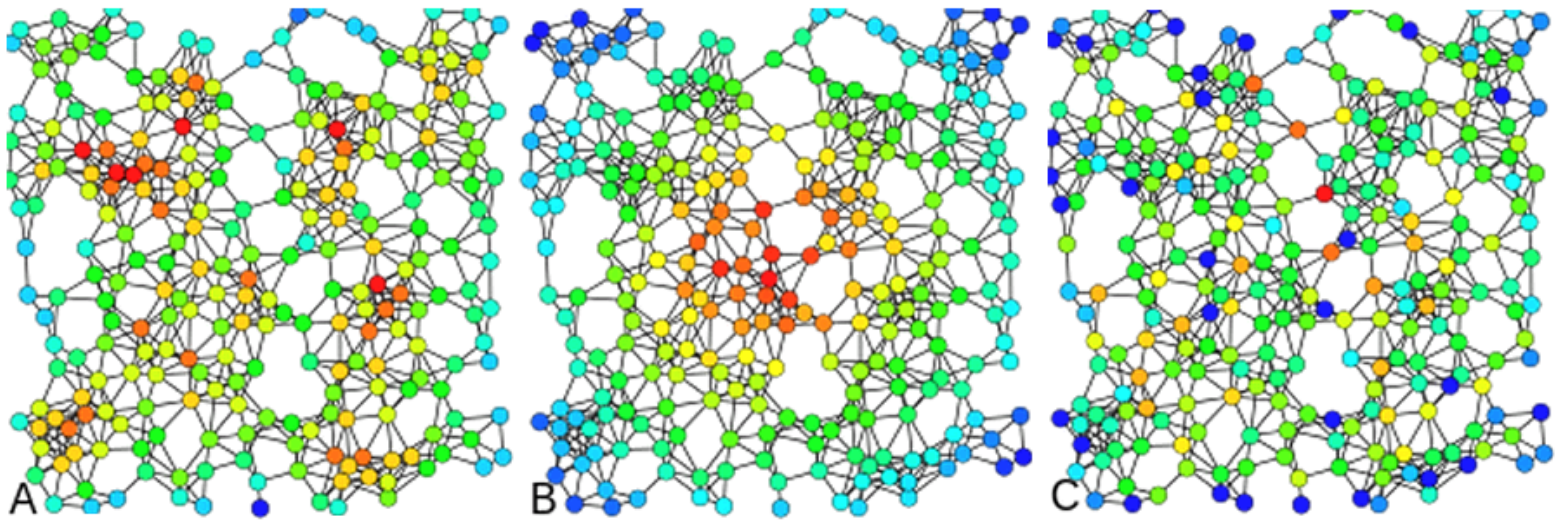


Visual Comparison

A Degree Centrality

B Closeness Centrality

C Betweenness Centrality



Axioms for centrality (Boldi and Vigna 2013)

Assessing

Question

Is there a robust way to convince oneself that a certain centrality measure is better than another?

Answer

Axiomatization...

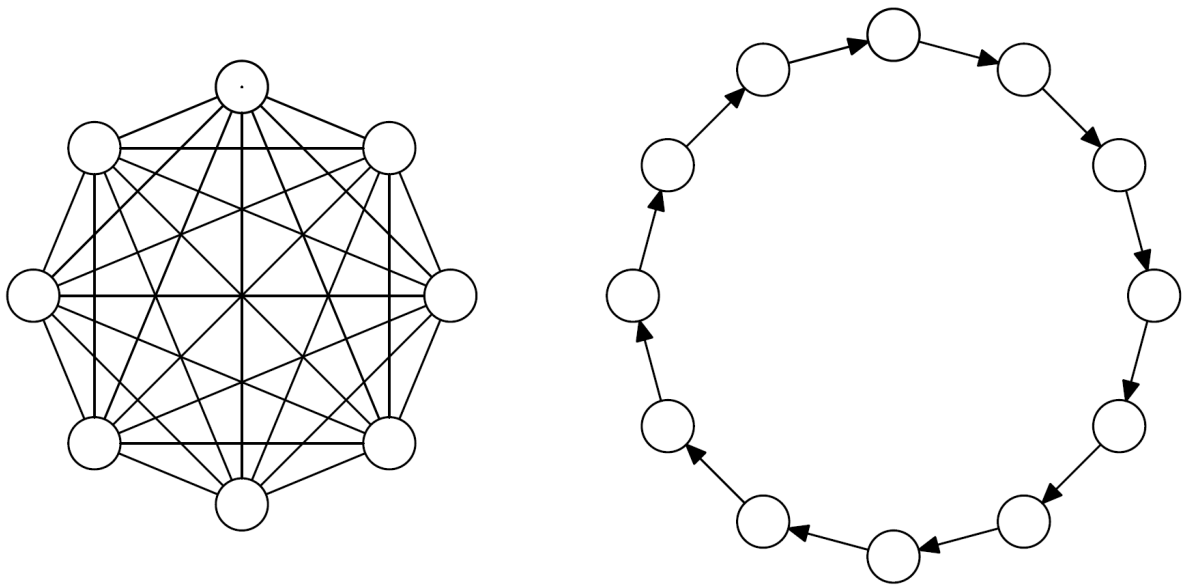
- ...hard axioms (characterize a centrality measure completely)
- ...soft axioms (like the T_i axioms for topological spaces)

Sensitivity to size

Idea: size matters!

$S_{k,p}$ be the union of a k-clique and a p-cycle.

- if $k \to \infty$, every vertex of the clique becomes ultimately strictly more important than every vertex of the cycle
- if $p \to \infty$, every vertex of the cycle becomes ultimately strictly more important than every vertex of the clique

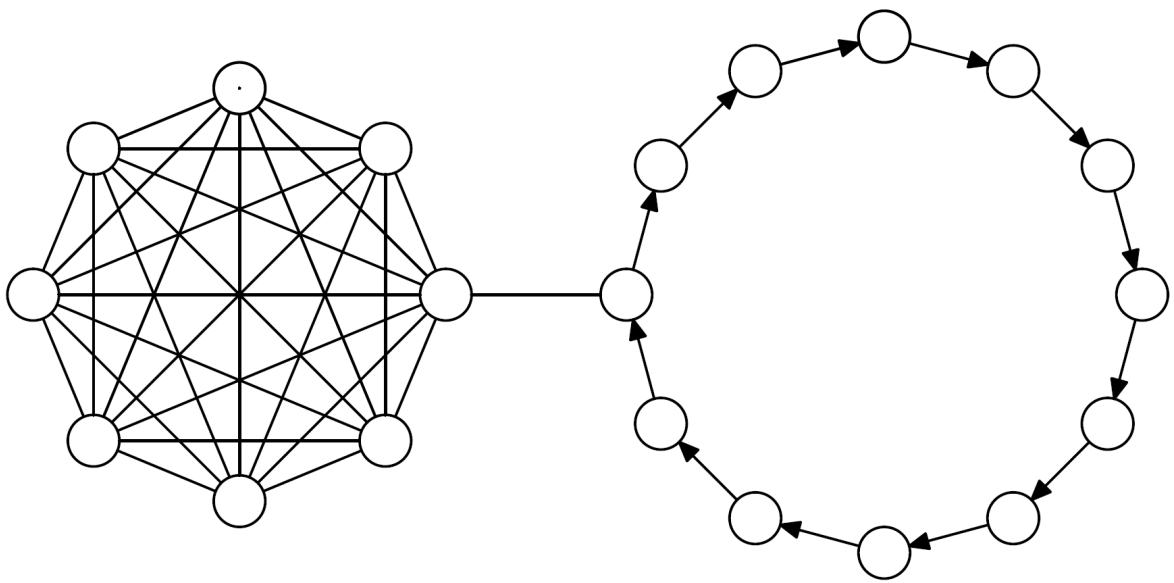


Sensitivity to density

Idea: density matters!

$D_{k,p}$ be made by a k-clique and a p-cycle connected by a single bidirectional bridge:

- if $k \to \infty$, the vertex on the clique-side of the bridge becomes more important than the vertex on the cycle-side.



Score monotonicity

Adding an edge $x \to y$ strictly increases the score of y.

Doesn't say anything about the score of other vertexes!

Rank monotonicity

Adding an edge $x \to y$...

- if y used to dominate z, then the same holds after adding the edge
- if y had the same score as z, then the same holds after adding the edge
- **strict variant:** if y had the same score as z, then y dominates z after adding the edge

Rank monotonicity

	Monotonicity				Other axioms	
	General		Strongly connected			
Centrality	Score	Rank	Score	Rank	Size	Density
Harmonic	yes	yes*	yes	yes*	yes	yes
Degree	yes	yes*	yes	yes*	only k	yes
Katz	yes	yes*	yes	yes*	only k	yes
PageRank	yes	yes*	yes	yes*	no	yes
Seeley	no	no	yes	yes	no	yes
Closeness	no	no	yes	yes	no	no
Lin	no	no	yes	yes	only k	no
Betweenness	no	no	no	no	only p	no
Dominant	no	no	?	?	only k	yes
HITS	no	no	no	no	only k	yes
SALSA	no	no	no	no	no	yes

Kendall's τ

Hollywood collaboration network

	degree	Katz 1/4λ	Katz 1/2λ	Katz 3/4λ	SALSA	closeness	harmonic	Lin	HITS	between	PR 1/4	PR 1/2	PR 3/4
degree	1.0000	0.9709	0.9287	0.8627	0.9005	0.4357	0.5526	0.5512	0.5170	0.5034	0.3699	0.4225	0.5074
Katz 1/4λ	0.9709	1.0000	0.9609	0.8957	0.8719	0.4638	0.5816	0.5801	0.5476	0.5026	0.3448	0.3964	0.4801
Katz 1/2λ	0.9287	0.9609	1.0000	0.9369	0.8291	0.4965	0.6157	0.6139	0.5849	0.4952	0.3108	0.3605	0.4416
Katz 3/4λ	0.8627	0.8957	0.9369	1.0000	0.7630	0.5488	0.6697	0.6676	0.6478	0.4811	0.2633	0.3098	0.3865
SALSA	0.9005	0.8719	0.8291	0.7630	1.0000	0.5371	0.4519	0.4504	0.4185	0.4692	0.4496	0.5042	0.5924
closeness	0.4357	0.4638	0.4965	0.5488	0.5371	1.0000	0.8503	0.8508	0.7366	0.3293	0.1529	0.1813	0.2319
harmonic	0.5526	0.5816	0.6157	0.6697	0.4519	0.8503	1.0000	0.9925	0.8694	0.3929	0.0752	0.1041	0.1549
Lin	0.5512	0.5801	0.6139	0.6676	0.4504	0.8508	0.9925	1.0000	0.8680	0.3916	0.0753	0.1041	0.1546
HITS	0.5170	0.5476	0.5849	0.6478	0.4185	0.7366	0.8694	0.8680	1.0000	0.3645	0.0518	0.0780	0.1249
between	0.5034	0.5026	0.4952	0.4811	0.4692	0.3293	0.3929	0.3916	0.3696	1.0000	0.4852	0.4909	0.4923
PR 1/4	0.3699	0.3448	0.3108	0.2633	0.4496	0.1529	0.0752	0.0753	0.0518	0.4852	1.0000	0.9317	0.8276
PR 1/2	0.4225	0.3964	0.3605	0.3098	0.5042	0.1813	0.1041	0.1041	0.0780	0.4909	0.9317	1.0000	0.8952
PR 3/4	0.5074	0.4801	0.4416	0.3865	0.5924	0.2319	0.1549	0.1546	0.1249	0.4923	0.8276	0.8952	1.0000

.uk (May 2007 snapshot)

	degree	Katz 1/4λ	Katz 1/2λ	Katz 3/4λ	SALSA	closeness	harmonic	Lin	HITS	PR 1/4	PR 1/2	PR 3/4
degree	1.0000	0.9053	0.9024	0.9000	0.9114	0.1950	0.2060	0.2060	0.2853	0.6449	0.6161	0.5784
Katz 1/4λ	0.9053	1.0000	0.9957	0.9922	0.8141	0.2059	0.2268	0.2265	0.2773	0.5917	0.5820	0.5595
Katz 1/2λ	0.9024	0.9957	1.0000	0.9966	0.8112	0.2078	0.2289	0.2286	0.2776	0.5914	0.5827	0.5611
Katz 3/4λ	0.9000	0.9922	0.9966	1.0000	0.8089	0.2094	0.2307	0.2303	0.2778	0.5911	0.5832	0.5622
SALSA	0.9114	0.8141	0.8112	0.8089	1.0000	0.1782	0.1617	0.1619	0.1917	0.6445	0.6146	0.5747
closeness	0.1950	0.2059	0.2078	0.2094	0.1782	1.0000	0.8592	0.8566	0.3817	0.1518	0.1746	0.2004
harmonic	0.2060	0.2268	0.2289	0.2307	0.1617	0.8592	1.0000	0.9694	0.4253	0.1503	0.1770	0.2072
Lin	0.2060	0.2265	0.2286	0.2303	0.1619	0.8566	0.9694	1.0000	0.4272	0.1503	0.1768	0.2069
HITS	0.2853	0.2773	0.2776	0.2778	0.1917	0.3817	0.4253	0.4272	1.0000	0.1529	0.1484	0.1415
PR 1/4	0.6449	0.5917	0.5914	0.5911	0.6445	0.1518	0.1503	0.1503	0.1529	1.0000	0.9182	0.8289
PR 1/2	0.6161	0.5820	0.5827	0.5832	0.6146	0.1746	0.1770	0.1768	0.1484	0.9182	1.0000	0.9088
PR 3/4	0.5784	0.5595	0.5611	0.5622	0.5747	0.2004	0.2072	0.2069	0.1415	0.8289	0.9088	1.0000

Correlation

- most geometric indices and HITS are rather correlated to one another;
- Katz, degree and SALSA are also highly correlated;
- PageRank stands alone in the first dataset, but it is correlated to degree, Katz, and SALSA in the second dataset;
- Betweenness is not correlated to anything in the first dataset, and could not be computed in the second dataset due to the size of the graph (106M vertices).

Exact Algorithms

Outline

1. Exact algorithms for static graphs
    1. the standard algorithm for closeness
    2. the standard algorithm for betweenness
    3. a faster betweenness algorithm through shattering and compression
    4. a GPU-Based algorithm for betweenness
2. Exact algorithms for dynamic graphs
    1. a dynamic algorithm for closeness
    2. four dynamic algorithms for betweenness
    3. a parallel streaming algorithm for betweenness

Exact Algorithms for Static Graphs

Exact Algorithm for Closeness Centrality

(folklore)

Exact Algorithm for Closeness

Recall the definition:

$$c(x) = \frac{1}{\sum_{y \neq x} d(x, y)}$$

Fastest known algorithm for closeness: All-Pairs Shortest Paths

- Runtime: $O(nm + n^2 \log n)$

Too slow for web-scale graphs!

- Later we'll discuss an approximation algorithm

A Faster Algorithm for Betweenness Centrality

U. Brandes

Journal of Mathematical Sociology (2001)

Why faster?

Let's take a step back. Recall the definition

$$\sum_{\substack{s \neq x \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(x)}{\sigma_{st}}$$

- σ_{st}: no. of $\mathcal{S}$ (SPs) from s to t
- $\sigma_{st}(x)$: no. of $\mathcal{S}$ from s to t that go through x

We could:

1. obtain all the σ_{st} and $\sigma_{st}(x)$ for all x, s, t via APSP; and then
2. perform the aggregation to obtain $\mathrm{b}(x)$ for all x.

The first step takes $O(nm + n^2 \log n)$, but the second step takes... $\Theta(n^3)$ (a sum of $O(n^2)$ terms for each of the n vertices).

Brandes' algorithm interleaves the SP computation with the aggregation, achieving runtime $O(nm + n^2 \log n)$

I.e., it is faster than the APSP approach

Dependencies

Define: Dependency of s on v:

$$\delta_s(v) = \sum_{t \neq s \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Hence:

$$\mathrm{b}(v) = \sum_{s \neq v} \delta_s(v)$$

Brandes proved that $\delta_s(v)$ obeys a recursive relation:

$$\delta_s(v) = \sum_{w : v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_s(w))$$

We can leverage this relation for efficient computation of betweenness

Recursive relation

Theorem (Simpler form)

If there is exactly one $\mathcal{S}$ from s to each t, then

$$\delta_s(v) = \sum_{w : v \in P_s(w)} (1 + \delta_s(w))$$

Proof sketch:

- The $\mathbb{S}_{\mathrm{DAG}}$ from s is a tree;
- Fix t. v is either on the single $\mathcal{S}$ from s to t or not.
- v lies on all and only the SPs to vertices w for which v is a predecessor (one $\mathcal{S}$ for each w) and the SPs that these lie on. Hence the thesis.

The general version must take into account that not all SPs from s to w go trough v.

Brandes' Algorithm

1. Initialize $\delta_s(v)$ to 0 for each v, s and $\mathrm{b}(w)$ to 0 for each w.
2. Iterate the following loop for each vertex s:
   1. Run Dijkstra's algorithm from s, keeping track of σ_{sv} for each encountered vertex v, and inserting the vertices in a max-heap H by distance from s;
   2. While H is not empty:
      1. Pop the max vertex t in H;
      2. For each $w \in P_s(t)$, increment $\delta_s(w)$ by $\frac{\sigma_{sw}}{\sigma_{st}}(1 + \delta_s(t))$;
      3. Increment $\mathrm{b}(t)$ by $\delta_s(t)$;

Shattering and Compressing Networks for Betweenness Centrality

A. E. Sarıyüce, E. Saule, K. Kaya, Ü. V. Çatalyürek

SDM '13: SIAM Conference on Data Mining

Intuition

Observations:

- There are vertices with predictable betweenness (e.g., 0, or equal to one of their neighbors). We can remove them from the graph (compression)
- Partitioning the (compressed) graph into small components allows for faster SP computation (shattering)

Idea: We can iteratively compress & shatter until we can't reduce the graph any more.

Only at this point we run (a modified) Brandes's algorithm and then aggregate the "partial" betweenness in different components.

Introductory definitions

- Graph $G = (V, E)$
- Induced graph by $V' \subseteq V$: $G_{V'} = (V', E' = V' \times V' \cap E)$
- Neighborhood of a vertex v: $\Gamma(v) = \{u \ : \ (v, u) \in E\}$
- Side vertex: a vertex v such that $G_{\Gamma(v)}$ is a clique
- Identical vertices: two vertices u and v such that either $\Gamma(u) = \Gamma(v)$ or $\Gamma(u) \cup \{u\} = \Gamma(v) \cup \{v\}$

Compression

Empirical / intuitive observations

- if v has degree 1, then $\mathrm{b}(v) = 0$
- if v is a side vertex, then $\mathrm{b}(v) = 0$
- if u and v are identical, then $\mathrm{b}(v) = \mathrm{b}(w)$

Compression:

- remove degree-1 vertices and side vertices; and
- merge identical vertices

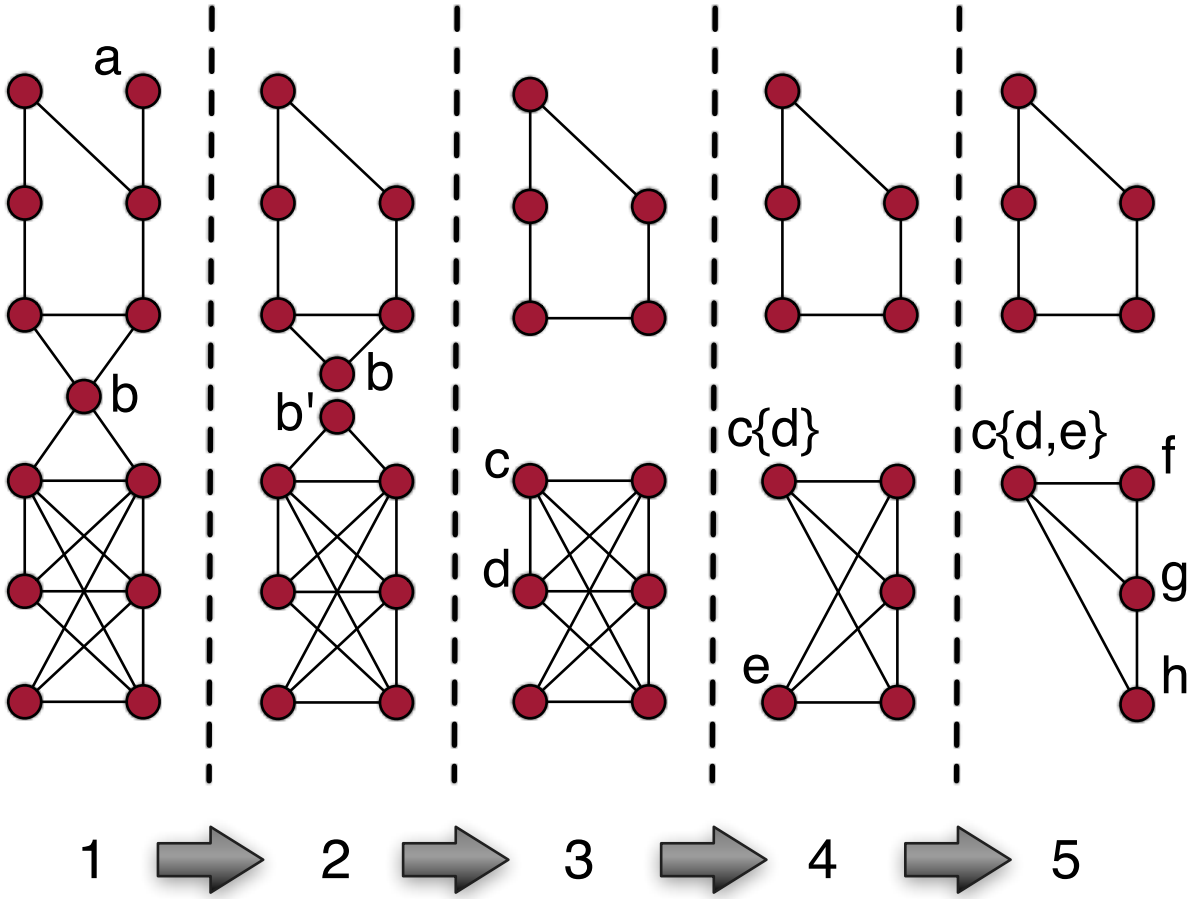
Shattering

- Articulation vertex: vertex v whose deletion makes the graph disconnected
- Bridge edge: an edge $e = (u, v)$ such that $G' = (V, E \setminus \{e\})$ has more components than G (u and v are articulation vertexes)

Shattering:

- remove bridge edges
- split articulation vertices in two copies, one per resulting component

Example of shattering and compression



Issues

Issues to take care of when iteratively compressing & shattering:

Example of issue

A vertex may have degree 1 only after we removed another vertex: we can't just remove and forget it, as its original betweenness was not 0.

Example of issue

When splitting an articulation vertex into component copies, we need to know, for each copy, how many vertices in other components are reachable through that vertex.

...and more

Solution

(Sketch)

- When we remove a vertex u, one of its neighbors (or an identical vertex) v is elected as the representative for u (and for all vertices that u was a representative of)
- We adjust the (current) values of $\mathrm{b}(v)$ and $\mathrm{b}(u)$ to appropriately take into account the removal of u

  the details are too hairy for a talk...
- When splitting articulation vertices or removing bridges, similar adjustments take place
- Brandes' algorithm is slightly modified to take the number of vertices that a vertex represents into consideration when computing the dependencies and the betweenness values

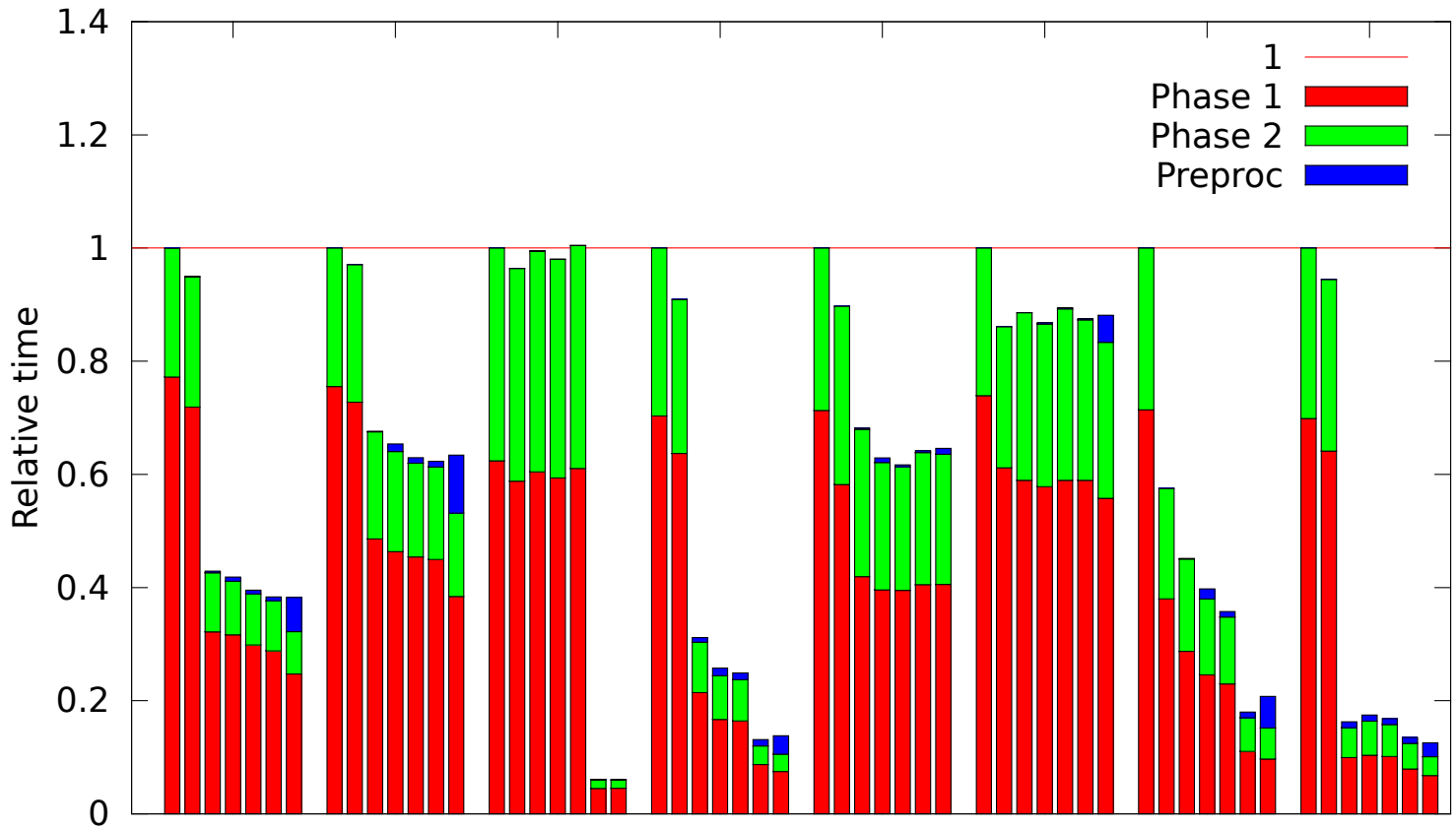
Speedup

"org." is Brandes' algorithm, "best" is compress & shatter

Graph			**Time** (in sec.)		
name	$\|V\|$	$\|E\|$	org.	best	Sp.
Power	4.9K	6.5K	1.47	0.60	2.4
Add32	4.9K	9.4K	1.50	0.19	7.6
HepTh	8.3K	15.7K	3.48	1.49	2.3
PGPgiant	10.6K	24.3K	10.99	1.55	7.0
ProtInt	9.6K	37.0K	11.76	7.33	1.6
AS0706	22.9K	48.4K	43.72	8.78	4.9
MemPlus	17.7K	54.1K	19.13	9.28	2.0
Luxemb.	114.5K	119.6K	771.47	444.98	1.7
AstroPh	16.7K	121.2K	40.56	19.41	2.0
Gnu31	62.5K	147.8K	422.09	188.14	2.2
CondM05	40.4K	175.6K	217.41	97.67	2.2
			geometric mean		**2.8**
Epinions	131K	711K	2,193	839	2.6
Gowalla	196K	950K	5,926	3,692	1.6
bcsstk32	44.6K	985K	687	41	16.5
NotreDame	325K	1,090K	7,365	965	7.6
RoadPA	1,088K	1,541K	116,412	71,792	1.6
Amazon0601	403K	2,443K	42,656	36,736	1.1
Google	875K	4,322K	153,274	27,581	5.5
WikiTalk	2,394K	4,659K	452,443	56,778	7.9
			geometric mean		**3.8**

Composition of runtime

- Preproc is the time needed to compress & shatter, Phase 1 is SSSP, Phase 2 is aggregation
- Different column for different variants of the algorithm (e.g., only compression of 1-degree vertices, only shattering of edges)
- the lower the better



Betweenness Centrality on GPUs and Heterogeneous Architectures

A. E. Sarıyüce, K. Kaya, E. Saule, Ü. V. Çatalyürek

GPGPU '13: Workshop on General Purpose Processing Using GPUs

Parallelism

- Fine grained: single concurrent BFS
- Only one copy of auxiliary data structures
- Synchronization needed
- Better for GPUs, which have small memory

- Coarse grained: many independent BFSs
- Sources are independent, embarrassingly parallel
- More memory needed
- Better for CPUs, which have large memory

GPU

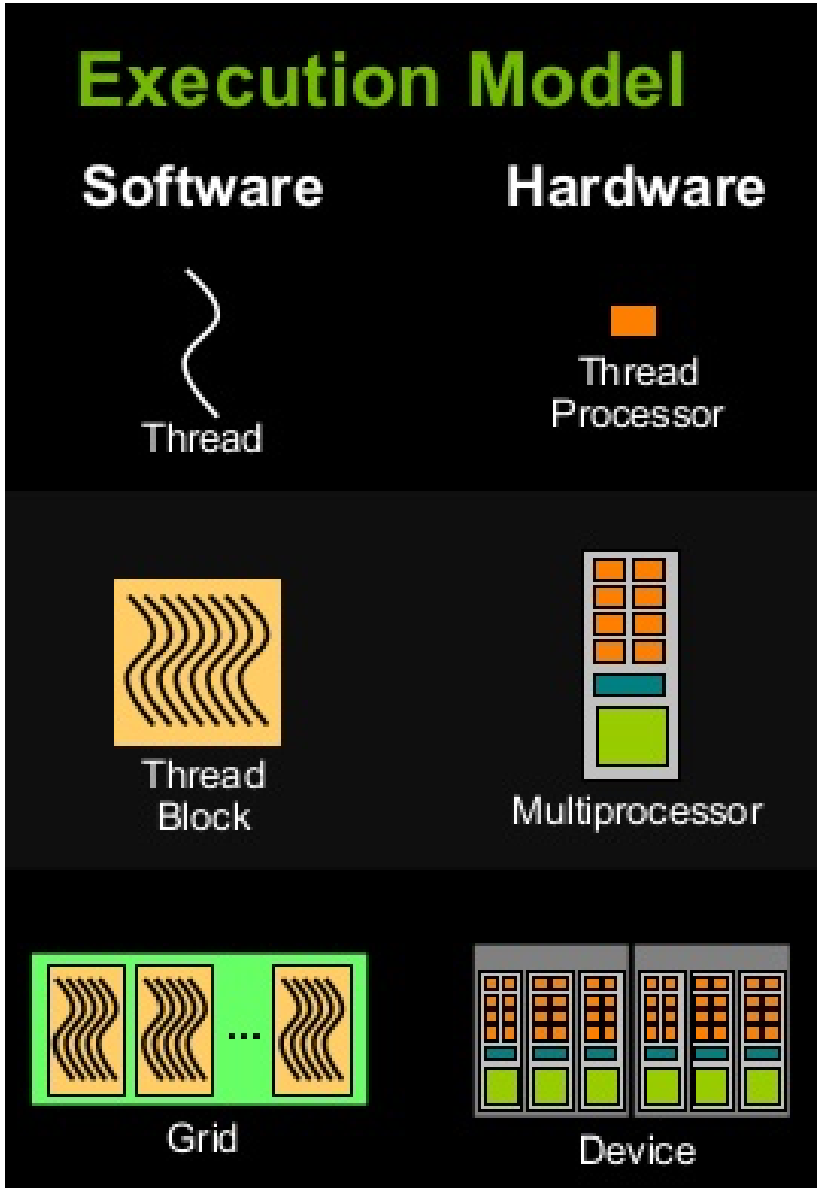
A GPU is especially well-suited to address problems that can be expressed as ***data-parallel computations*** *- the same program is executed on many data elements in parallel - with* ***high arithmetic intensity*** *- the ratio of arithmetic operations to memory operations.*
Because the same program is executed for each data element, there is a lower requirement for sophisticated flow control, and because it is executed on many data elements and has high arithmetic intensity, the memory access latency can be hidden with calculations instead of big data caches.[1]

[1] docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

Execution model

- One thread per data element
- Thread scheduled in blocks with barriers (wait for others at the end)
- Program runs on the whole data (kernel)

- Minimize synchronization
- Balance load
- Coalesce memory access

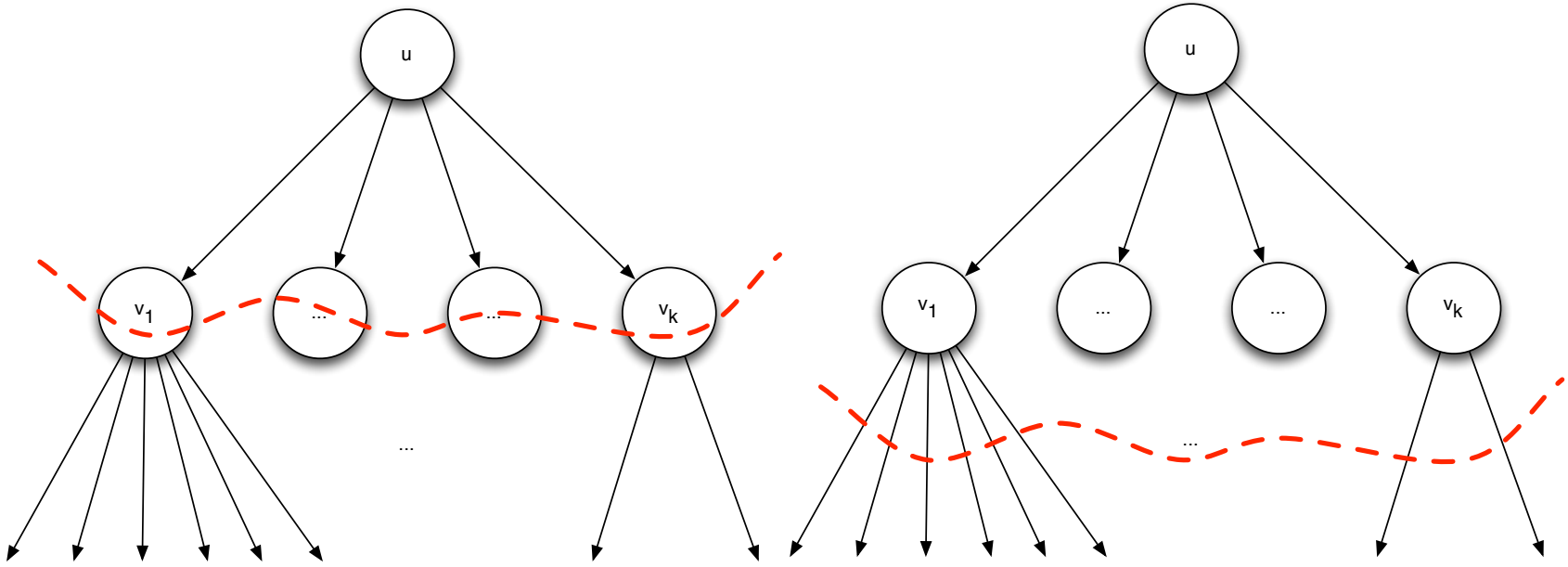


Intuition

- GPUs have huge number of cores
- Use them to parallelize BFS
- One core per vertex, or one core per edge
- Vertex-based parallelism creates load imbalance for graphs with skewed degree distribution
- Edge-based parallelism requires high memory usage

- Use vertex-based parallelism
- Virtualize high-degree vertices to address load imbalance
- Reduce memory usage by removing predecessors lists

Difference



Vertex-based BFS

Edge-based BFS

Vertex-based

- For each level, for each vertex in parallel
- If vertex is on level
- For each neighbor, adjust P and σ
- Atomic update on σ needed (multiple paths can be discovered concurrently)
- While backtracking, if $u \in P(v)$ accumulate $\delta(u) = \delta(u) + \delta(v)$
- Possible load imbalance if degree skewed

Algorithm 2: VERTEX: vertex-based parallel BC

```
...
ℓ ← 0
▷Forward phase
while cont = true do
    cont ← false
    ▷Forward-step kernel
    for each u ∈ V in parallel do
1       if d[u] = ℓ then
2           for each v ∈ Γ(u) do
3               if d[v] = −1 then
                    d[v] ← ℓ + 1, cont ← true
                else if d[v] = ℓ − 1 then P_v[u] ← 1
4               if d[v] = ℓ + 1 then σ[v] ←(atomic) σ[v] + σ[u]
    ℓ ← ℓ + 1
...
▷Backward phase
while ℓ > 1 do
    ℓ ← ℓ − 1
    ▷Backward-step kernel
    for each u ∈ V in parallel do
        if d[u] = ℓ then
5           for each v ∈ Γ(u) do
6               if P_v[u] = 1 then δ[u] ← δ[u] + δ[v]
▷Update bc values by using Equation (5)
...
```

Edge-based

- For each level, for each edge in parallel
- If edge endpoint is on level
- Same as above...
- While backtracking, if $u \in P(v)$ accumulate $\delta(u) = \delta(u) + \delta(v)$ atomically
- Multiple edges can try to update δ concurrently
- More memory (edge-based layout) and more atomic operations

Algorithm 3: EDGE: edge-based parallel BC

```
...
ℓ ← 0
▷Forward phase
while cont = true do
    cont ← false
    ▷Forward-step kernel
    for each (u, v) ∈ E in parallel do
1       if d[u] = ℓ then
            ... ▷same as vertex-based forward step
    ℓ ← ℓ + 1
...
▷Backward phase
while ℓ > 1 do
    ℓ ← ℓ − 1
    ▷Backward-step kernel
    for each (u, v) ∈ E in parallel do
        if d[u] = ℓ then
2           if P_v[u] = 1 then δ[u] <-atomic- δ[u] + δ[v]
▷Update bc values by using Equation (5)
...
```

Vertex virtualization

- AKA, edge batching, hybrid between vertex- and edge-based
- Split high degree vertices into virtual ones with maximum degree *mdeg*
- Equivalently, pack up to *mdeg* edges belonging to the same vertex together
- Very small $mdeg = 4$
- Need additional auxiliary maps

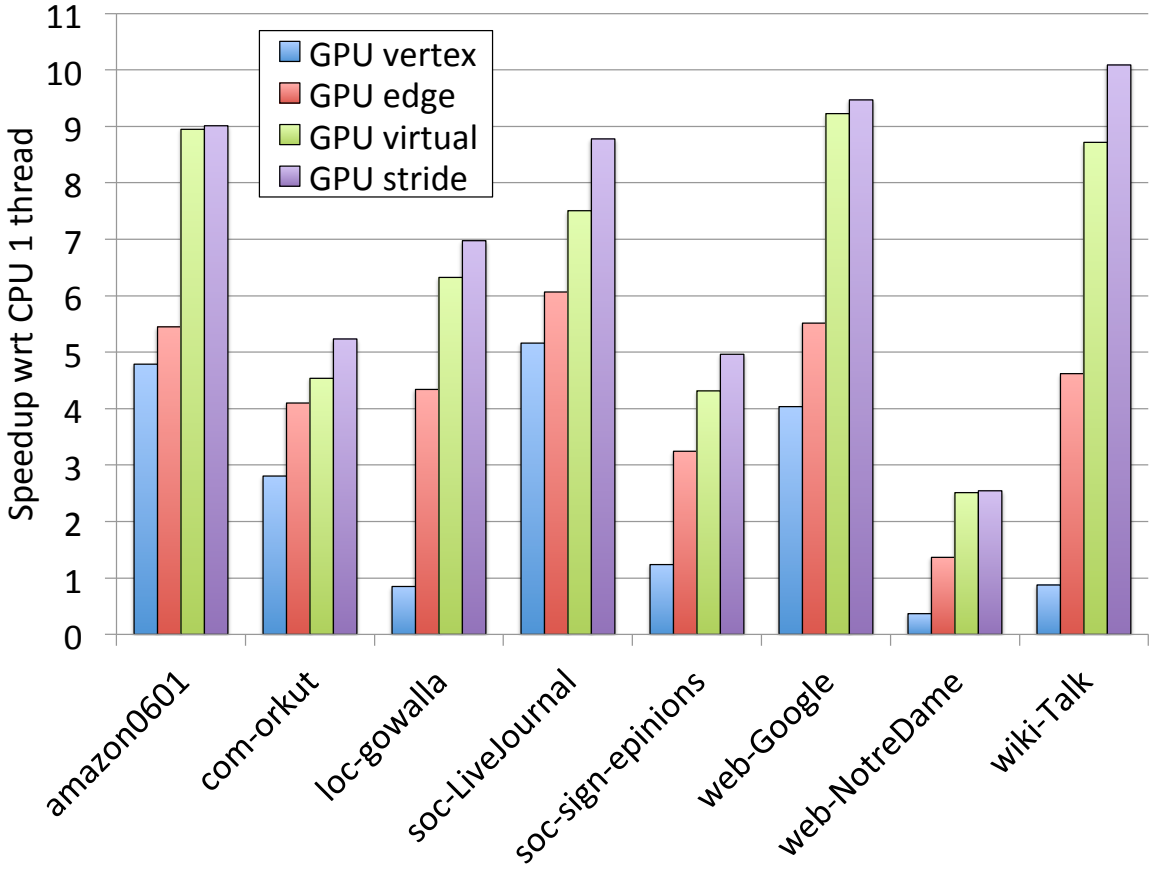
Algorithm 4: VIRTUAL: BC with virtual vertices

```
   ...
   ℓ ← 0
   ▷Forward phase
   while cont = true do
   |   cont ← false
   |   ▷Forward-step kernel
   |   for each virtual vertex u_vir in parallel do
   |   |   u ← vmap[u_vir]
   |   |   if d[u] = ℓ then
1  |   |   |   for each v ∈ Γ_vir(u_vir) do
2  |   |   |   |   if d[v] = −1 then
   |   |   |   |   |   d[v] ← ℓ + 1, cont ← true
3  |   |   |   |   if d[v] = ℓ + 1 then  σ[v] ←(atomic) σ[v] + σ[u]
   |   ℓ ← ℓ + 1
   ...
   ▷Backward phase
   while ℓ > 1 do
   |   ℓ ← ℓ − 1
   |   ▷Backward-step kernel
   |   for each virtual vertex u_vir in parallel do
   |   |   u ← vmap[u_vir]
   |   |   if d[u] = ℓ then
   |   |   |   sum ← 0
4  |   |   |   for each v ∈ Γ(u) do
5  |   |   |   |   if d[v] = ℓ + 1 then  sum ← sum + δ[v]
6  |   |   |   δ[u] ←(atomic) δ[u] + sum
   ▷Update bc values by using Equation (5)
   ...
```

Benefits

- Compared to vertex-based:
  - Reduce load imbalance
- Compared to edge-based:
  - Reduce number of atomic operations
  - Reduce memory footprint
- Predecessors stored implicitly in the $\mathbb{S}_{\text{DAG}}$ level (reduced memory usage)
- Memory layout can be further optimized to coalesce latency via striding:
  - Distribute edges to virtual vertices in round-robin
  - When accessed in parallel, they create faster sequential memory access pattern

Results



Speedup over Brandes' on CPU on real graphs with 32-core GPU ($s = 1k, \ldots, 100k$)

- Results computed only on a sample of sources and extrapolated linearly

Exact Algorithms for Dynamic Graphs

A Fast Algorithm for Streaming Betweenness Centrality

O. Green, R. McColl, D. A. Bader

SocialCom '12: International Conference on Social Computing

Intuition

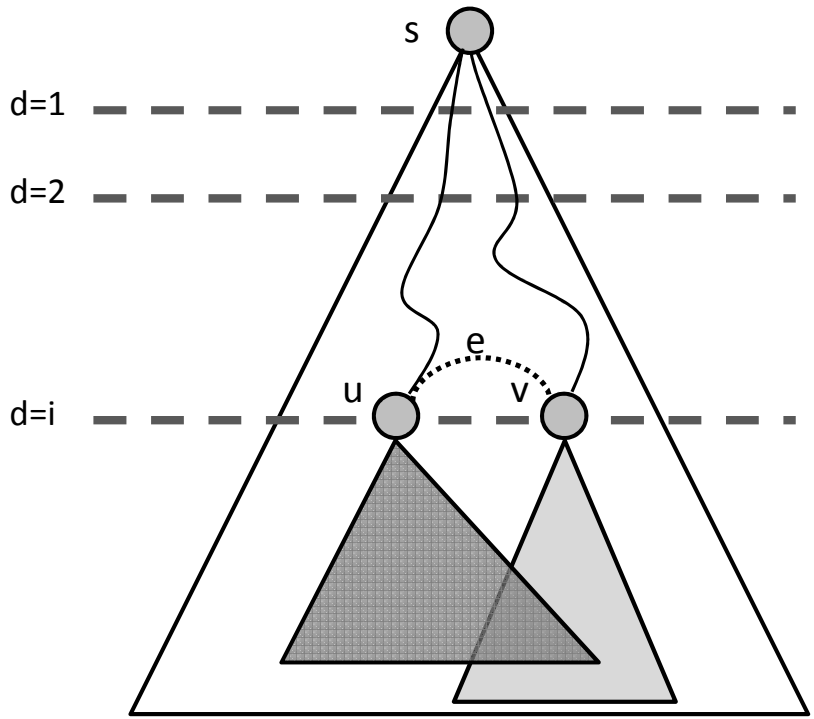
- Make Brandes' algorithm incremental
- Keep additional data structures to avoid recomputing partial results
  - Rooted $\mathbb{S}_{\mathrm{DAG}}$ for each source $s \in V$
  - Depth in the tree for t = distance of t from s
- Re-run parts of modified Brandes' algorithm on edge update
- Support only edge addition (on unweighted graphs)

Data structures

- One $\mathbb{S}\text{DAG}_s$ for each source $s \in V$, which contains for each other vertex $t \in V$:
  - Distance d_{st}, paths σ_{st}, dependencies $\delta_s(t)$, predecessors $P_s(t)$
  - Additional per-level queues for exploration
- On addition of edge (u, v), let $dd = |d_{su} - d_{sv}|$:
  - $dd = 0$ same level
  - $dd = 1$ adjacent level
  - $dd > 1$ non-adjacent level

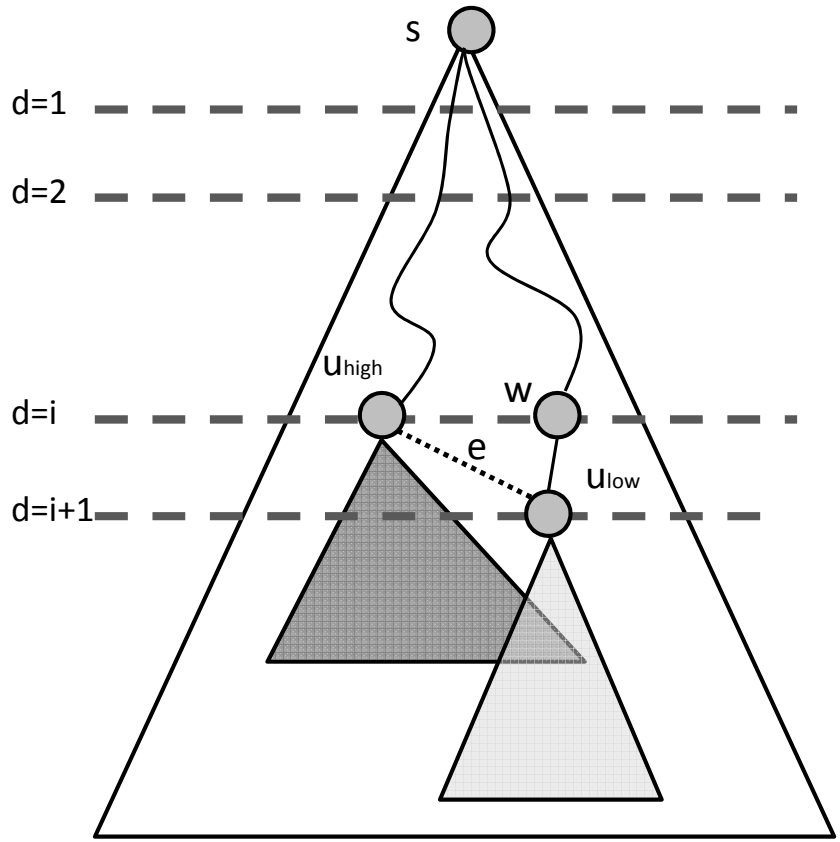
Same level addition

- $dd = 0$
- Edge creates no new shortest paths
- No change to betweenness due to this source



Adjacent level addition

- $dd = 1$
- Let $u_{high} = u$, $u_{low} = v$
- Edge creates new shortest paths
- $\mathbb{S}_{\mathrm{DAG}}$ unchanged
- Changes in σ confined to sub-dag rooted in u_{low}
- Changes in δ also spread above to decrease old dependency and account for new dependency
- Example: w and predecessors have now only $1/2$ of dependency on sub-dag rooted in u_{low}



Algorithm

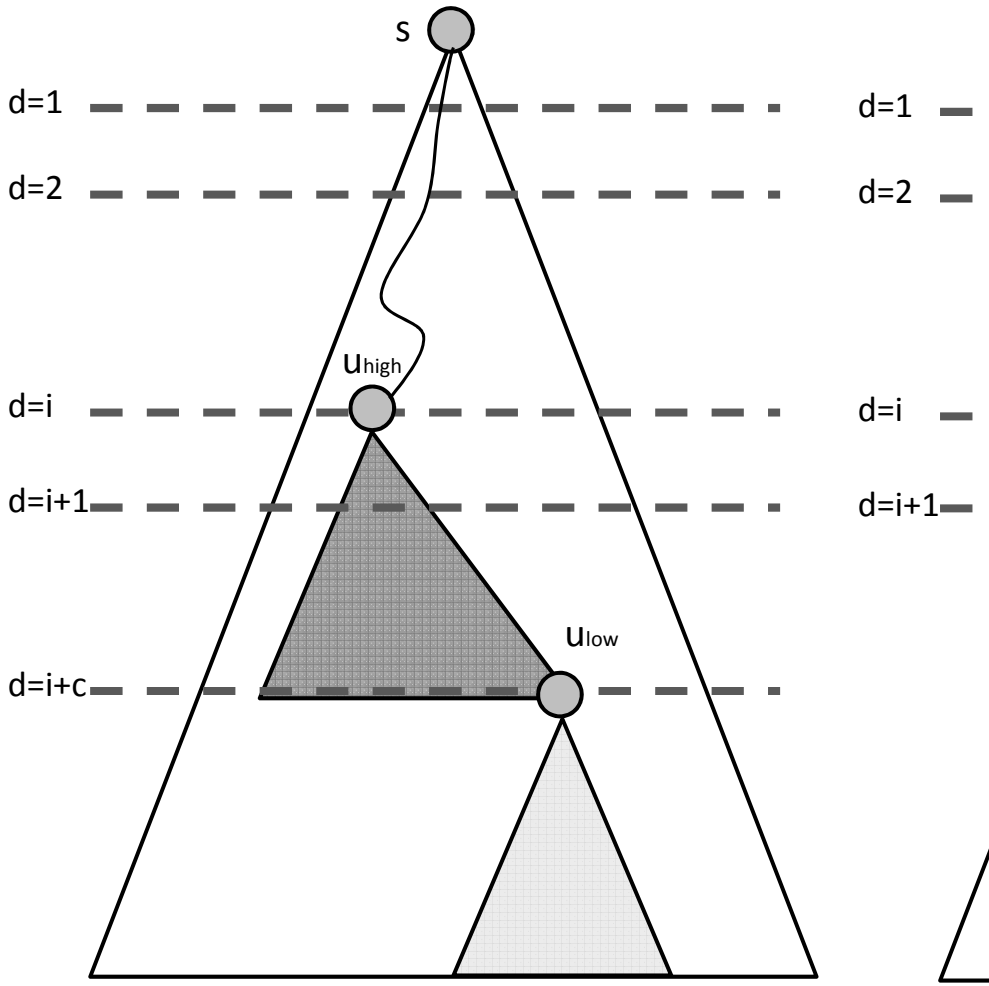
- During exploration:
  - Fix σ
  - Mark visited vertices
  - Enqueue for further processing
- During backtracking:
  - Fix δ and b
  - Recurse up the whole $\mathbb{S}_{\text{DAG}}$

Stage 2 - BFS traversal starting at u_{low}
while Q *not empty* **do**
  dequeue $v \leftarrow Q$;
  for *all neighbor* w *of* v **do**
    if $d[w] = (d[v] + 1)$ **then**
      if $t[w] =$ *Not-Touched* **then**
        enqueue $w \rightarrow Q_{BFS}$;
        enqueue $w \rightarrow Q[d[w]]$;
        $t[w] \leftarrow$ Down;
        $d[w] \leftarrow d[v] + 1$;
        $dP[w] \leftarrow dP[v]$;
      else
        $dP[w] \leftarrow dP[w] + dP[v]$;
      $\hat{\sigma}[w] \leftarrow \hat{\sigma}[w] + dP[v]$;

Stage 3 - modified dependency accumulation
$\hat{\delta[v]} \leftarrow 0, v \in \forall V$; $level \leftarrow V$;
while *level>0* **do**
  while $Q[level]$ *not empty* **do**
    dequeue $w \leftarrow Q[level]$;
    for *all* $v \in P[w]$ **do**
      if $t[v] =$*Not-Touched* **then**
        enqueue $v \rightarrow Q[level - 1]$;
        $t[v] \leftarrow$ Up;
        $\hat{\delta}[v] \leftarrow \delta[v]$;
      $\hat{\delta}[v] \leftarrow \hat{\delta}[v] + \frac{\hat{\sigma}[v]}{\hat{\sigma}[w]}(1 + \hat{\delta}[w])$;
      if $t[v] = Up \wedge (v \neq u_{high} \vee w \neq u_{low})$ **then**
        $\hat{\delta}[v] \leftarrow \hat{\delta}[v] - \frac{\sigma[v]}{\sigma[w]}(1 + \delta[w])$;
      if $w \neq r$ **then**
        $C_B[w] \leftarrow C_B[w] + \hat{\delta}[w] - \delta[w]$;
  $level \leftarrow level - 1$;
$\sigma[v] \leftarrow \hat{\sigma}[v], v \in \forall V$;

Non-adjacent level addition

- $dd > 1$
- Edge creates new shortest paths
- Changes to $\mathbb{S}_{\mathrm{DAG}}$ (new distances)
- Algorithm only sketched (most details missing)

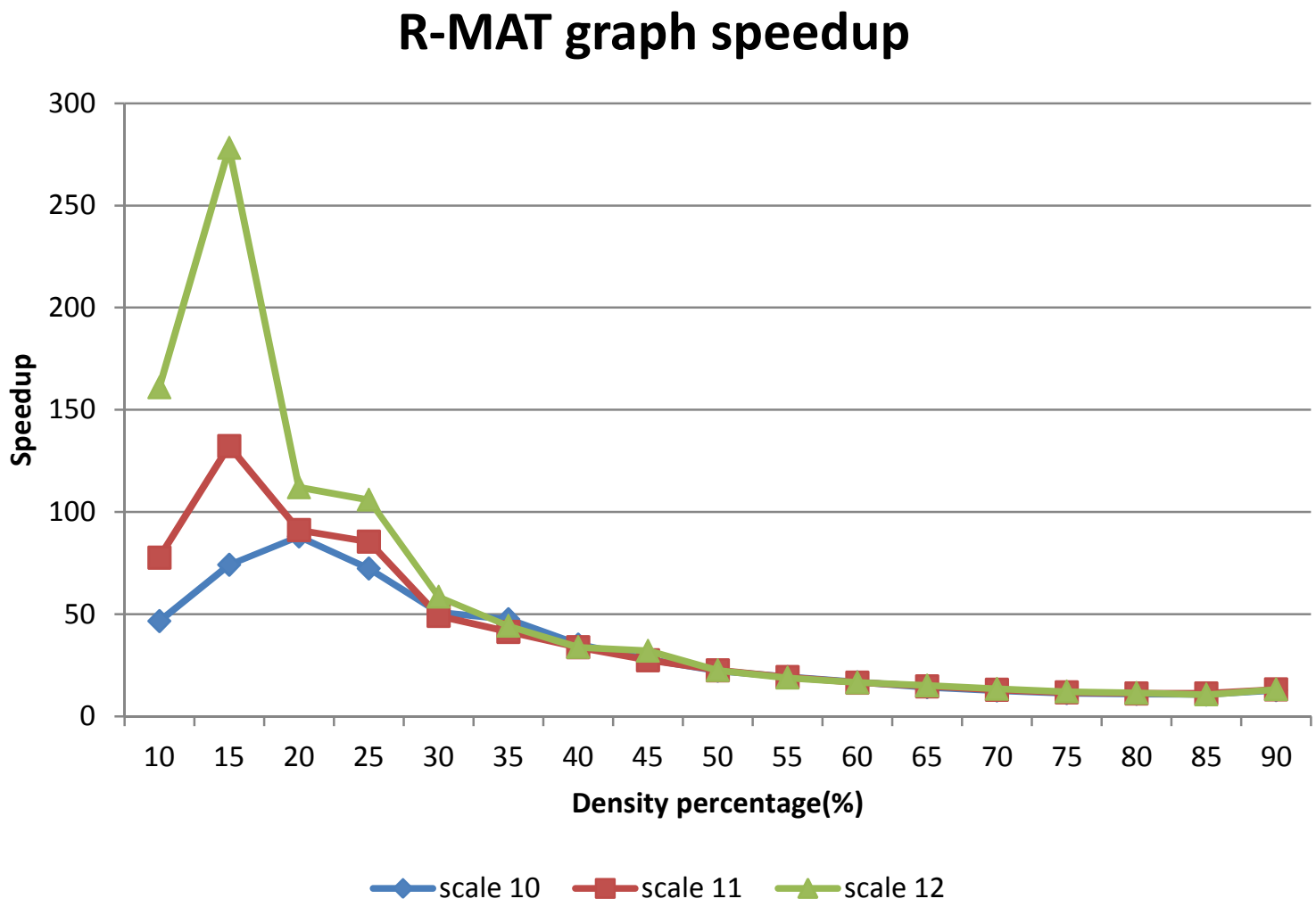


Complexity

- Time: $O(n^2 + nm) \leftarrow$ same as Brandes'
- In practice, algorithm is much faster

- Space: $O(n^2 + nm) \leftarrow$ higher than Brandes'
- For each source, a $\mathbb{S}\text{DAG}$ of complexity $n + m$

Results



Speedup over Brandes' on synthetic graphs ($n = 4096$)

Conclusions

- Up to 2 orders of magnitude speedup
- Super-quadratic space bottleneck

QUBE: a Quick algorithm for Updating BEtweenness centrality

M. Lee, J. Lee, J. Park, R. Choi, C. Chung

WWW '12: International World Wide Web Conference

Intuition

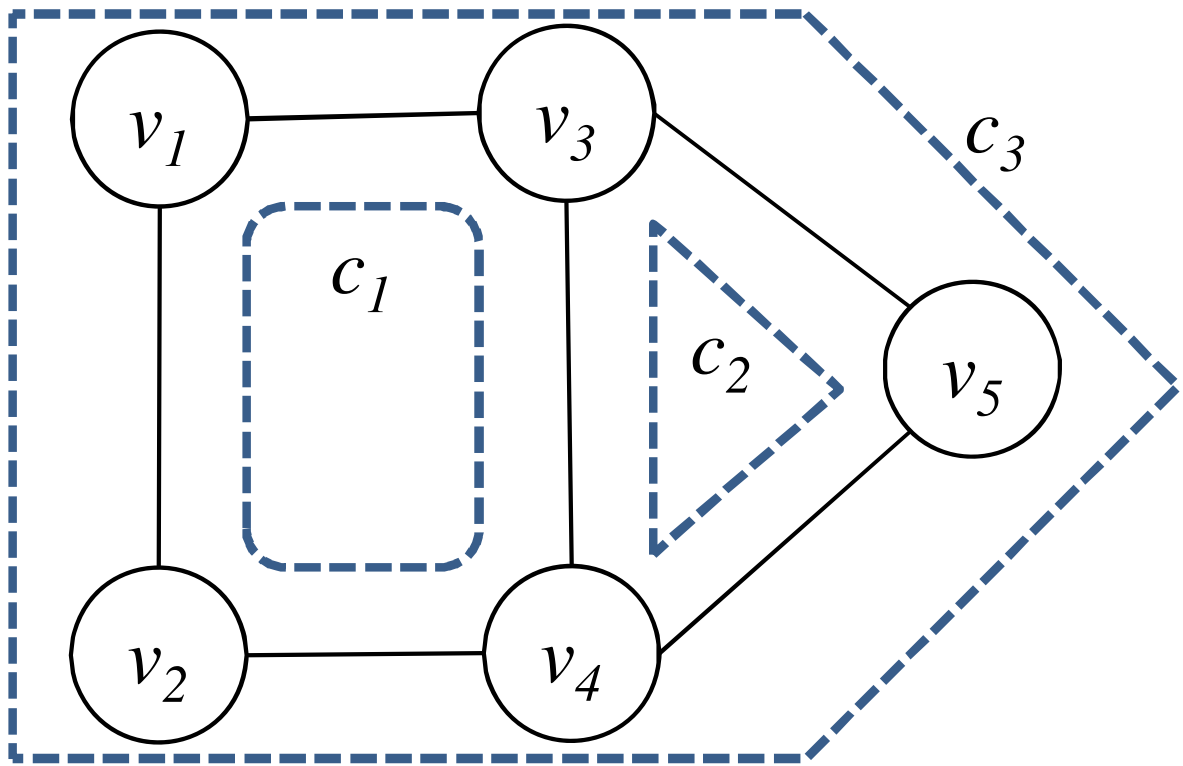
- No need to update all vertices when a new edge is added
- Prune vertices whose b does not change
- Large reduction in all-pairs shortest paths to be re-computed
- Support both edge additions and removals

Minimum Cycle Basis

- $G = (V, E)$ undirected graph
- Cycle $C \subseteq E$ s.t. $\forall v \in V$, v incident to even number of edges in C
- Represented as edge incidence vector $\nu \in \{0,1\}^{|E|}$, where $\nu(e) = 1 \iff e \in C$
- Cycle Basis = set of linearly independent cycles
- Minimum Cycle Basis = on weighted graph with non-negative weights w_e, cycle basis of minimum total weight $w(C) = \sum_i w(C_i)$ where $w(C_i) = \sum_{e \in C_i} w_e$

Minimum Cycle Basis Example

- Three cycle basis sets: $\{C_1, C_2\}, \{C_1, C_3\}, \{C_2, C_3\}$
- If all edges have same weight $w_e = 1$, $MCB = \{C_1, C_2\}$



Minimum Union Cycle

- Given a MCB C and minimum cycles $C_i \in C$
- Let V_{C_i} be the set of vertices induced by C_i
- Recursively union two V_{C_i} if they share at least one vertex
- The final set of vertices is a Minimum Union Cycle MUC

- MUCs are disjoint sets of vertices
- $MUC(v) =$ the MUC which contains vertex v

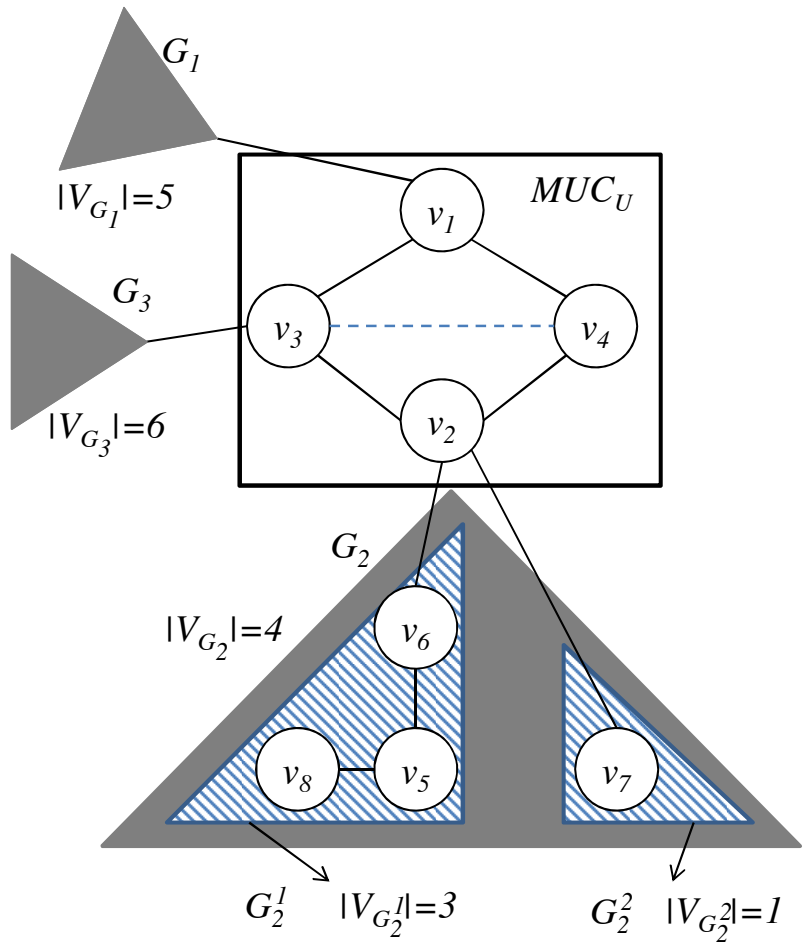
Connection Vertex

- Articulation Vertex = vertex v whose deletion makes the graph disconnected
- Biconnected graph = graph with no articulation vertex
- Vertex v is an articulation vertex $\iff$ v belongs to two biconnected components

- Connection Vertex = vertex v that
  - is an articulation vertex
  - has an edge to vertex $w \notin MUC(v)$

Connection Vertex Example

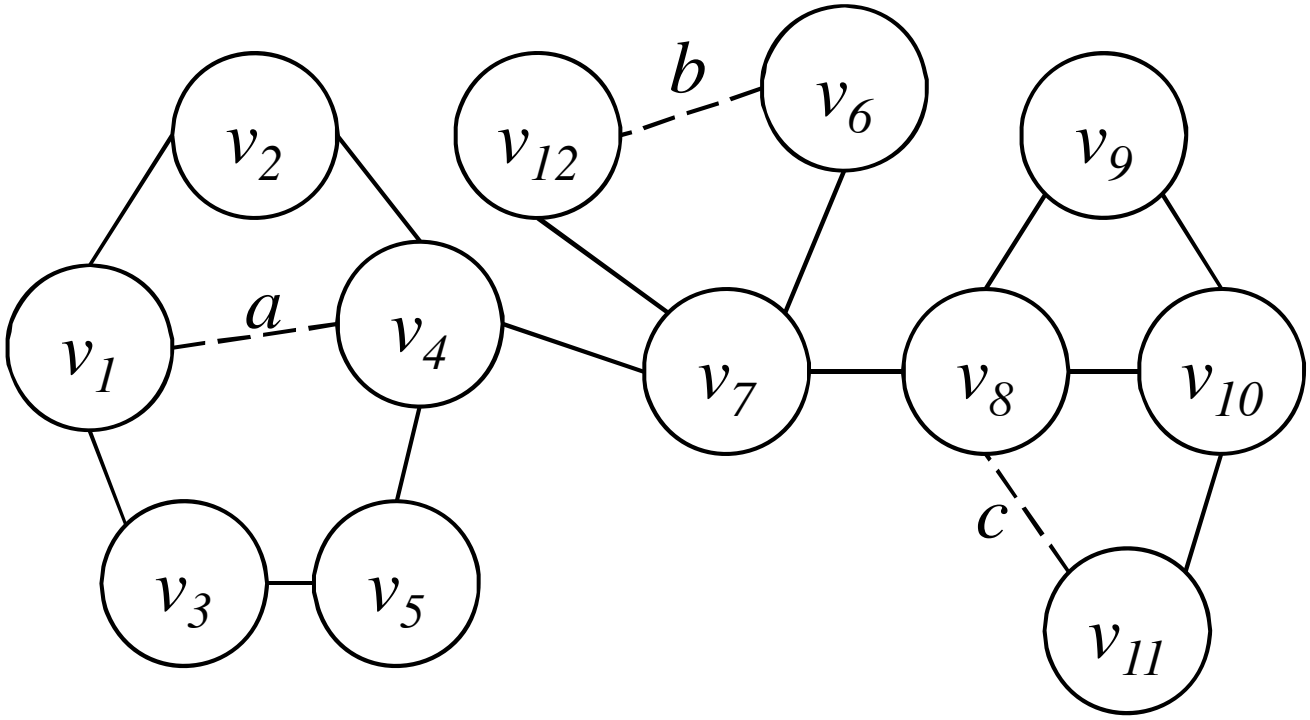
- If (v_3, v_4) is added, $MUC(v_3) = \{v_1, v_2, v_3, v_4\}$
- v_1, v_2, v_3 are connection vertices of $MUC(v_3)$
- Let G_i be the disconnected subgraph generated by removing v_i



Finding MUCs

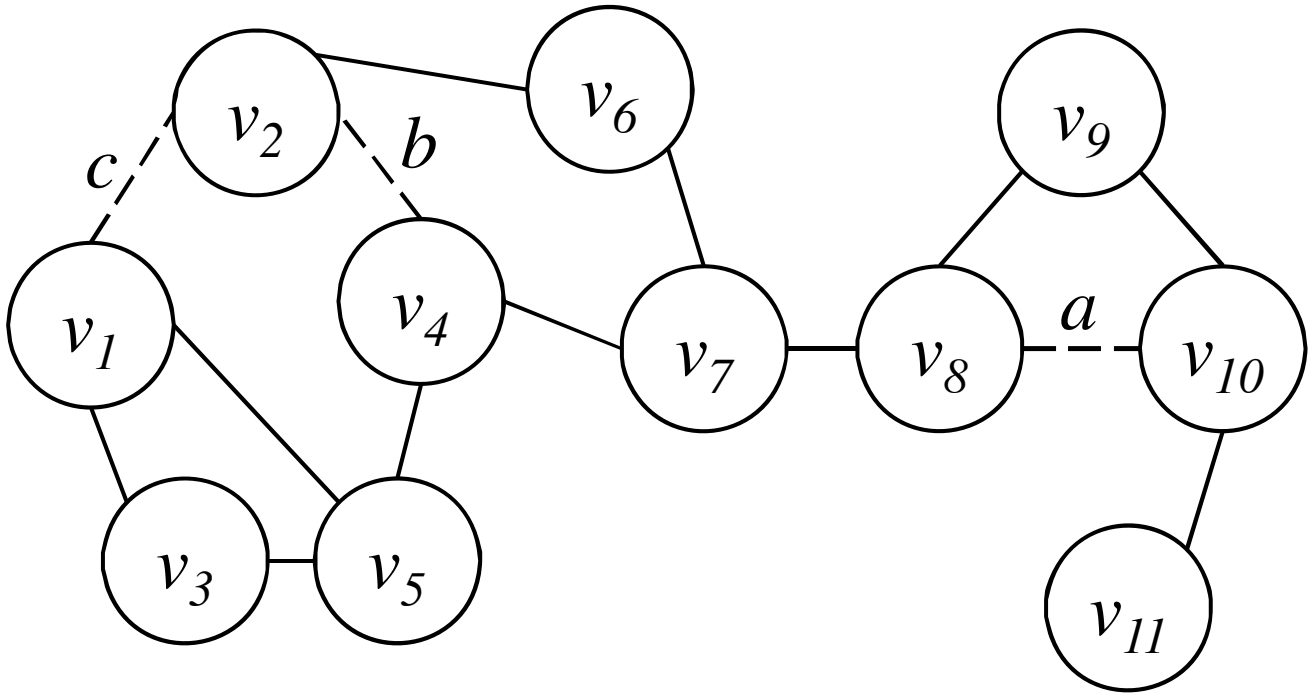
- Finding an *MCB* is well studied
- Kavitha, Mehlhorn, Michail, Paluch. "A faster algorithm for minimum cycle basis of graphs". ICALP 2004
- Finding *MUC* from *MCB* relatively straightforward (just union sets of vertices)
- Also find connection vertices for each *MUC*
- All done as a preprocessing step
- Need to be updated at runtime

Updating MUCs – Addition



- Adding a does not affect the MUC (endpoints in the same MUC)
- Adding b creates a new MUC (endpoints do not belong to a MUC)
- Adding c merges two MUCs (merge MUCs of vertices on the $\mathcal{S}$ between endpoints)

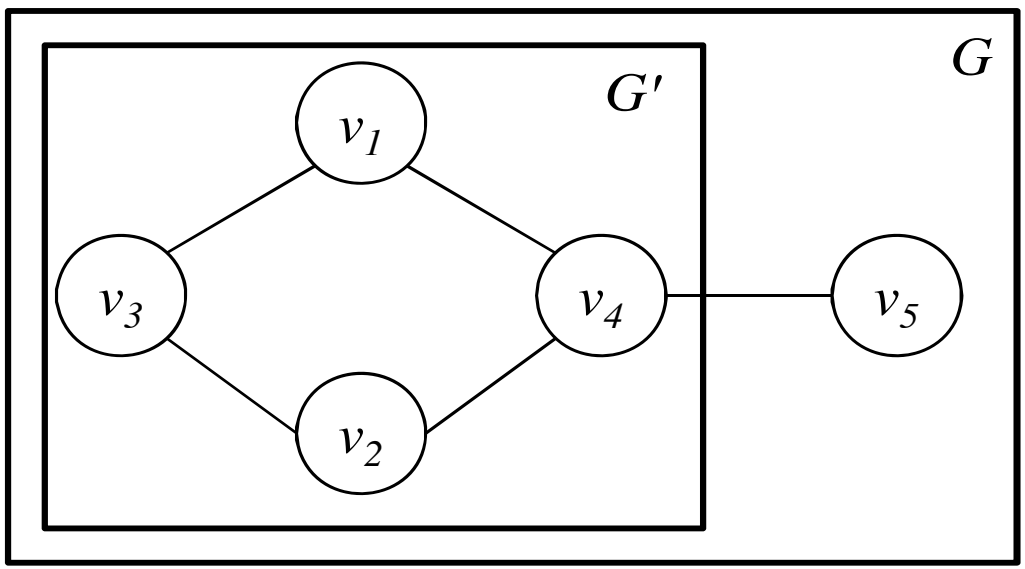
Updating MUCs – Removal



- Removing a destroys the MUC (cycle is removed $\rightarrow$ no biconnected component)
- Removing b does not affect the MUC (MUC is still biconnected)
- Removing c splits the MUC in two (single vertex appears in all $\mathcal{S}$ between endpoints)

Betweenness Centrality Dependency

- Only vertexes inside the *MUC*s of the updated endpoints need to be updated
- However, recomputing all centralities for the *MUC* still requires new shortest paths to the rest of the graph
  - Shortest paths to vertices outside the *MUC*
  - Shortest paths that pass through the *MUC*



	G	G'
$c(v_1)$	1	0.5
$c(v_2)$	1	0.5
$c(v_3)$	0.5	0.5
$c(v_4)$	3.5	0.5
$c(v_5)$	0	

Betweenness Centrality outside the MUC

- Let $s \in V_{G_j}$, $t \in MUC$,
- Let $j \in MUC$ be a connection vertex to subgraph G_j
- Each vertex in $\mathcal{S}_{jt}$ is also in $\mathcal{S}_{st}$
- Therefore, betweenness centrality due to vertices outside the MUC:

$$b_o(v) = \begin{cases} \frac{|V_{G_j}|}{\sigma_{st}} & \text{if } v \in \{\mathcal{S}_{jt} \setminus t\} \\ 0 & \text{otherwise} \end{cases}$$

Betweenness Centrality trough the MUC

- Let $s \in V_{G_j}$, $t \in V_{G_k}$,
- Let $j \in MUC$ be a connection vertex to subgraph G_j
- Let $k \in MUC$ be a connection vertex to subgraph G_k
- Each vertex in $\mathcal{S}_{jk}$ is also in $\mathcal{S}_{st}$
- Therefore, betweenness centrality due to paths through the MUC:

$$b_x(v) = \begin{cases} \frac{|V_{G_j}||V_{G_k}|}{\sigma_{st}} & \text{if } v \in \mathcal{S}_{jk} \\ 0 & \text{otherwise} \end{cases}$$

More caveats apply for subgraphs that are disconnected, as every path that connects vertices in different connected component passes through v

Updating Betweenness Centrality

$$\mathrm{b}(v) = \mathrm{b}_{MUC}(v) + \sum_{G_j \subset G} \mathrm{b}_o(v) + \sum_{G_j, G_k \subset G} \mathrm{b}_x(v)$$

QUBE algorithm

Algorithm 3: QUBE(MUC_U)

input : MUC_U - Minimum Union Cycle that updated vertices belong to
output : $C[v_i]$ - Updated Betweenness Centrality Array

```
1 begin
2 |   Let SP be the set of all pair shortest paths in MUC_U ;
3 |   Let C[v_i] be an empty array, v_i ∈ MUC_U ;
4 |   SP, C[v_i] ← Betweenness() ;
5 |   for each shortest path <v_a, ..., v_b> in SP do
6 |   |   if v_a is a connecting vertex then
7 |   |   |   G_a := Subgraph connected by a connection
  |   |   |   vertex v_a ;
```

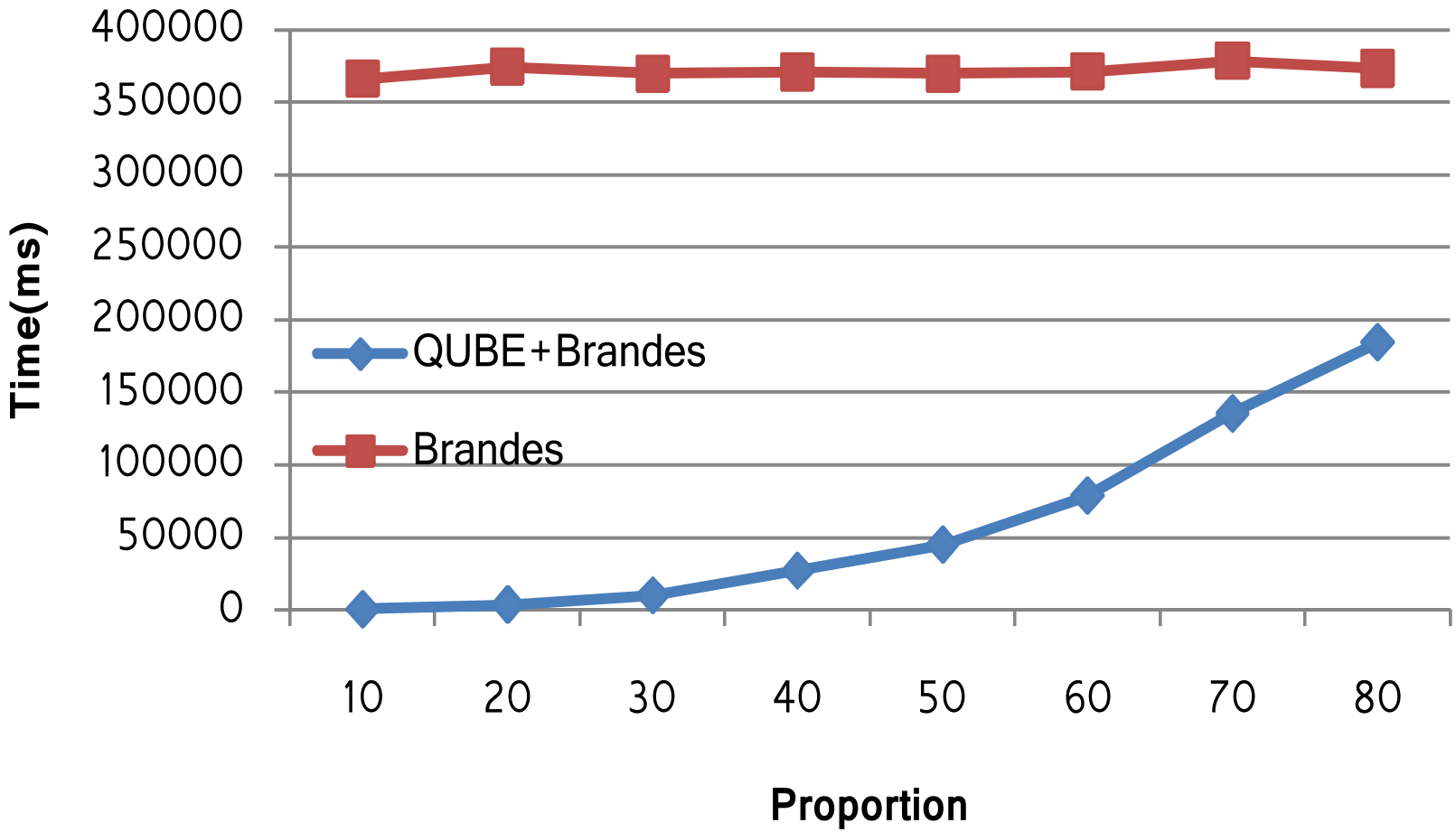
QUBE algorithm

```
8   for each v_i ∈ <v_a, ..., v_b> - {v_b} do
9       C[v_i] := C[v_i] + |V_{G_a}| / |SP(v_a, v_b)| ;
10      if v_b is also a connecting vertex then
11          G_b := Subgraph connected by a
                   connection vertex v_b ;
12          for each v_i ∈ < v_a, ..., v_b > do
13              C[v_i] := C[v_i] + |V_{G_a}|·|V_{G_b}| / |SP(v_a, v_b)| ;
14  if G_a is disconnected then
15      C[v_a] := C[v_a] + |V_{G_a}|^2 − Σ_{l=1}^{n} (|V_{G_a^l}|^2)
```

QUBE + Brandes

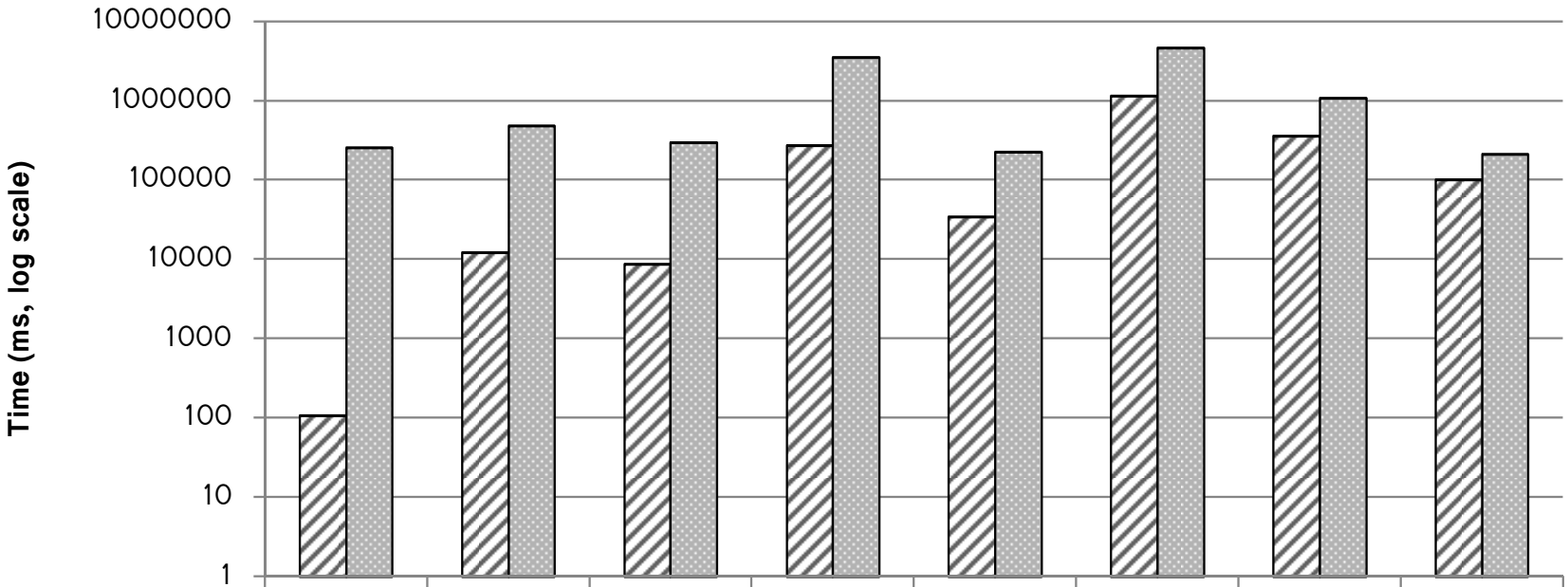
- QUBE is a pruning rule that reduces the search space for betweenness recomputation
- Can be paired with any existing betweenness algorithm to compute b_{MUC}
- In the experiments, Brandes' is used
- Quantities computed by Brandes' (e.g., σ) reused by QUBE for b_o and b_x

Results



Update time as a function of the percentage of vertices of the graph in the updated *MUC* for synthetic Erdös-Rényi graphs ($n = 5000$)

Conclusions



	Eva	Erdos02	Erdos972	Pgp	Epa	Contact	Wikivote	CAGrQc
QUBE+Brandes	106	12289	8640	270419	34056	1150801	361362	101895
Brandes	256326	486267	297100	3538417	227158	4600805	1082843	210831

- Improvement depends highly on structure of the graph (bi-connectedness)
- From 2 orders of magnitude (best) to 2 times (worst) faster than Brandes'

Incremental Algorithm for Updating Betweenness Centrality in Dynamically Growing Networks

M. Kas, M. Wachs, K. M. Carley, L. R. Carley

ASONAM '13: International Conference on Advances in Social Networks analysis and Mining

Intuition

- Extend an existing dynamic all-pairs shortest path algorithm to betweenness
- G. Ramalingam and T. Reps, "On the Computational Complexity of Incremental Algorithms," CS, Univ. of Wisconsin at Madison, Tech. Report 1991
- Relevant quantities: number of shortest paths σ, distances d, predecessors P
- Keep a copy of the old quantities while updating
- Support only edge addition (on weighted graphs)

Edge update

- Compute new shortest paths from updated endpoints (u, v)
- If a new shortest path of the same length is found, updated number of paths as

$$\sigma_{st} = \sigma_{st} + \sigma_{su} \times \sigma_{vt}$$

- If a new shorter shortest path to any vertex is found, update d, clear σ
- Betweenness decreased if new shortest path found
- Edge betweenness updates backtrack via DFS over $P_s(t)$

$$\mathrm{b}(w) = \mathrm{b}(w) - \sigma_{sw} \times \sigma_{wt} / \sigma_{st}$$

Edge update

- Complex bookkeeping: need to consider all affected vertices which have new alternative shortest paths of equal length (not covered in the original algorithm)
- Amend P during update propagation $\rightarrow$ concurrent changes to the $\mathbb{S}$DAG
- Need to track now-unreachable vertices separately

- After having fixed d, σ, b, increase b due to new paths
- Update needed $\forall s, t \in V$ affected by changes (tracked from previous phase)
- Betweenness increase analogous to above decrease

Results

Network	D?	#(N)	#(E)	Avg Speedup	Affect%
SocioPatterns	U	113	4392	9.58 x	38.26%
FB-like	D	1896	20289	18.48 x	27.67%
HEP Coauthor	U	7507	19398	357.96 x	42.08%
P2P Comm.	D	6843	7572	36732 x	0.02%

Speedup over Brandes' on real-world graphs

- Speedup depends on topological characteristics (e.g., diameter, clust. coeff.)

Comparison with QUBE

Network	Type	#(Node)	#(Edge)	QuBE	Incremental Betweenness
Eva [24]	Ownership	4457	4562	2418.17	25425.87
CAGrQc [25]	Collaboration	4158	13422	2.06	67.86

Speedup over Brandes' in comparison with QUBE

- Datasets from the QUBE paper
- About 1 order of magnitude faster than QUBE

Betweenness Centrality – Incremental and Faster

M. Nasre, M. Pontecorvi, V. Ramachandran

MFCS '14: Mathematical Foundations of Computer Science

Intuition

- Keep $\mathbb{S}_{\text{DAG}}$ for each vertex
- Re-use information from $\mathbb{S}_{\text{DAG}}$ of updated edge endpoints
- Adding new edges will not make old edges part of a $\mathcal{S}$
- Support only edge addition (on weighted graphs)

Main Result

- Let $E^* = \bigcup_{e \in \mathbb{S}} e \subseteq E$ be the set of edges that are part of any shortest path
- Let $m^* = |E^*|$ and $\nu^* = \max_{v \in V} |\mathbb{S}_{\mathrm{DAG}_v}|$ the maximum number of edges in shortest paths through any single vertex v
- $n < \nu^* < m^* < m$
- After incremental update, betweenness can be recomputed in
  - $O(\nu^* n)$ time using $O(\nu^* n)$ space
  - $O(m^* n)$ time using $O(n^2)$ space
- Bounded by $O(mn + n^2)$
- Logarithmic factor better than Brandes' (on weighted graphs)

Lemma 1

Lemma 1. *If weight of edge (u, v) in G is decreased to obtain G', then for any $x \in V$, the set of shortest paths from x to u and from v to x is the same in G and G', and $d'(x, u) = d(x, u)$, $d'(v, x) = d(v, x)$; $\sigma'_{xu} = \sigma_{xu}$, $\sigma'_{vx} = \sigma_{vx}$.*

- Edge $(u, v) \notin \mathcal{S}_{xu} \wedge (u, v) \notin \mathcal{S}_{vx}$ as edge weights are positive

Lemma 2

Lemma 2. *Let the weight of edge (u, v) be decreased to $\mathbf{w}'(u, v)$, and for any given pair of vertices s, t, let $D(s, t) = d(s, u) + \mathbf{w}'(u, v) + d(v, t)$. Then,*

1. *If $d(s, t) < D(s, t)$, then $d'(s, t) = d(s, t)$ and $\sigma'_{st} = \sigma_{st}$.*
   The shortest paths from s to t in G' are the same as in G.
2. *If $d(s, t) = D(s, t)$, then $d'(s, t) = d(s, t)$ and $\sigma'_{st} = \sigma_{st} + (\sigma_{su} \cdot \sigma_{vt})$.*
   The shortest paths from s to t in G' are a superset of the shortest paths G.
3. *If $d(s, t) > D(s, t)$, then $d'(s, t) = D(s, t)$ and $\sigma'_{st} = \sigma_{su} \cdot \sigma_{vt}$.*
   The shortest paths from s to t in G' are new (shorter distance).

- Updates to σ and d in constant time
- Need to update P to complete $\mathbb{S}$DAG update

$\mathbb{S}$DAG Update

Algorithm 3. Update-DAG$(s, \mathbf{w}'(u,v))$

Input: DAG(s), DAG(v), and $flag(s,t), \forall t \in V$.
Output: An edge set H after decrease of weight on edge (u,v), and $P'_s(t), \forall t \in V - \{s\}$.

1: $H \leftarrow \emptyset$.
2: **for** each $v \in V$ **do** $P'_s(v) = \emptyset$.
3: **for** each edge $(a,b) \in$ DAG(s) and $(a,b) \neq (u,v)$ **do**
4: **if** $flag(s,b) =$ UN-changed or $flag(s,b) =$ NUM-changed **then**
5: $H \leftarrow H \cup \{(a,b)\}$ and $P'_s(b) \leftarrow P'_s(b) \cup \{a\}$.
6: **for** each edge $(a,b) \in$ DAG(v) **do**
7: **if** $flag(s,b) =$ NUM-changed or $flag(s,b) =$ WT-changed **then**
8: $H \leftarrow H \cup \{(a,b)\}$ and $P'_s(b) \leftarrow P'_s(b) \cup \{a\}$.
9: **if** $flag(s,v) =$ NUM-changed or $flag(s,v) =$ WT-changed **then**
10: $H \leftarrow H \cup \{(u,v)\}$ and $P'_s(v) \leftarrow P'_s(v) \cup \{u\}$.

- UN-changed $\rightarrow dd = 0$
- NUM-changed $\rightarrow dd = 1$
- WT-changed $\rightarrow dd > 1$

Edge Update

Algorithm 4. Edge-Update($G = (V, E), \mathbf{w}'(u, v)$)

Input: updated edge with $\mathbf{w}'(u, v)$, $d(s, t)$ and σ_{st}, $\forall\ s, t \in V$; DAG$(s), \forall\ s \in V$.
Output: BC$'(v)$, $\forall\ v \in V$; $d'(s, t)$ and σ'_{st} $\forall\ s, t \in V$; DAG$'(s), \forall\ s \in V$.

1: **for** every $v \in V$ **do** BC$'(v) \leftarrow 0$.
   for every $s, t \in V$ **do** compute $d'(s, t), \sigma'_{st}$, $flag(s, t)$. // use Lemma 2
2: **for** every $s \in V$ **do**
3: Update-DAG$(s, (u, v))$. // use Alg. 3
4: Stack $S \leftarrow$ vertices in V in a reverse topological order in DAG$'(s)$.
5: Accumulate-dependency(s, S). // use Alg. 2

Space-Efficient Variant $O(n^2)$

- Do not store the $\mathbb{S}_{\mathrm{DAG}}$
- Store only E^*
- Updated $\mathbb{S}_{\mathrm{DAG}}$ can be build in $O(m^*)$ time
  - Time $O(m^* n)$
  - Compute E'^* from E^*, then $\mathbb{S}'_{\mathrm{DAG}s}$ from E'^*
- Space $O(m^* + n^2)$ to store E^* and n^2 distances $d(s,t)$ and shortest paths σ_{st}

Comparison

Paper	**Year**	**Space**	**Time**	**Weights**	**Update Type**
Brandes static [3]	2001	$O(m+n)$	$O(mn)$	NO	Static Alg.
Lee et al. [21]	2012	$O(n^2+m)$	Heuristic	NO	Single Edge
Green et al. [12]	2012	$O(n^2+mn)$	$O(mn)$	NO	Single Edge
Kourtellis+ [19]	2014	$O(n^2)$	$O(mn)$	NO	Single Edge
Singh et al. [10]	2013	–	Heuristic	NO	Vertex update
Brandes static [3]	2001	$O(m+n)$	$O(mn+n^2 \log n)$	YES	Static Alg.
Kas et al. [16]	2013	$O(n^2+mn)$	Heuristic	YES	Single Edge
This paper	2014	$O(\nu^* \cdot n)$	$O(\nu^* \cdot n)$	YES	Vertex Update
This paper	2014	$O(n^2)$	$O(m^* \cdot n)$	YES	Vertex Update

Conclusions

- Provably faster than Brandes' on weighted graphs
- However m^* can be large in practice
- No experiments
- Hard to parallelize (need to access pairs of $\mathbb{S}_{\text{DAG}}$ at a time)
- Still has main bottleneck of most algorithms: $O(n^2)$ memory

Incremental Algorithms for Closeness Centrality

A. E. Sarıyüce, K. Kaya, E. Saule, U. V. Çatalyürek

IEEE BigData '13: International Conference on Big Data

Intuition

- Algorithm with pruning based on level difference (similar to Green et al.)
- Additional pruning by bi-connected decomposition (similar to QUBE)
- Applied to closeness centrality (still solves APSP)
- Reminder: closeness centrality
- $c(v) = \frac{1}{\sum_{u \in V} d(u, v)}$

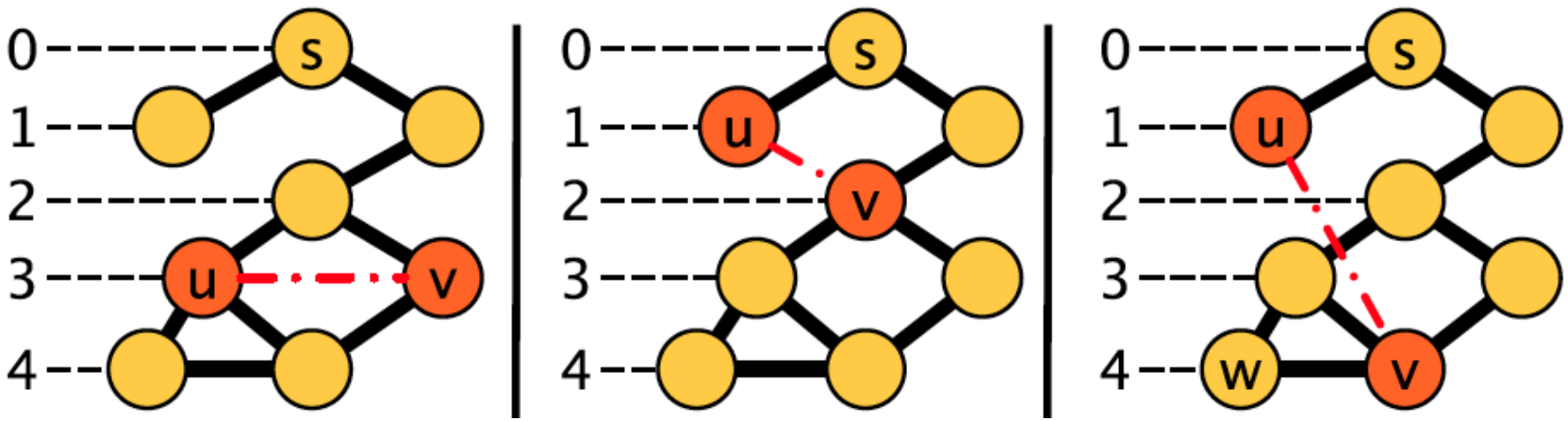
Preliminaries

- Best static algorithm $O(nm)$ time

Algorithm 1: CC: Basic centrality computation

Data: $G = (V, E)$
Output: $cc[.]$
1 **for each** $s \in V$ **do**
  ▷SSSP(G, s) with centrality computation
  $Q \leftarrow$ empty queue
  $d[v] \leftarrow \infty, \forall v \in V \setminus \{s\}$
  Q.push(s), $d[s] \leftarrow 0$
  $\texttt{far}[s] \leftarrow 0$
  while Q *is not empty* **do**
    $v \leftarrow Q$.pop()
    for all $w \in \Gamma_G(v)$ **do**
      if $d[w] = \infty$ **then**
        Q.push(w)
        $d[w] \leftarrow d[v] + 1$
        $\texttt{far}[s] \leftarrow \texttt{far}[s] + d[w]$
  $\texttt{cc}[s] = \frac{1}{\texttt{far}[s]}$
return cc[.]

Cases



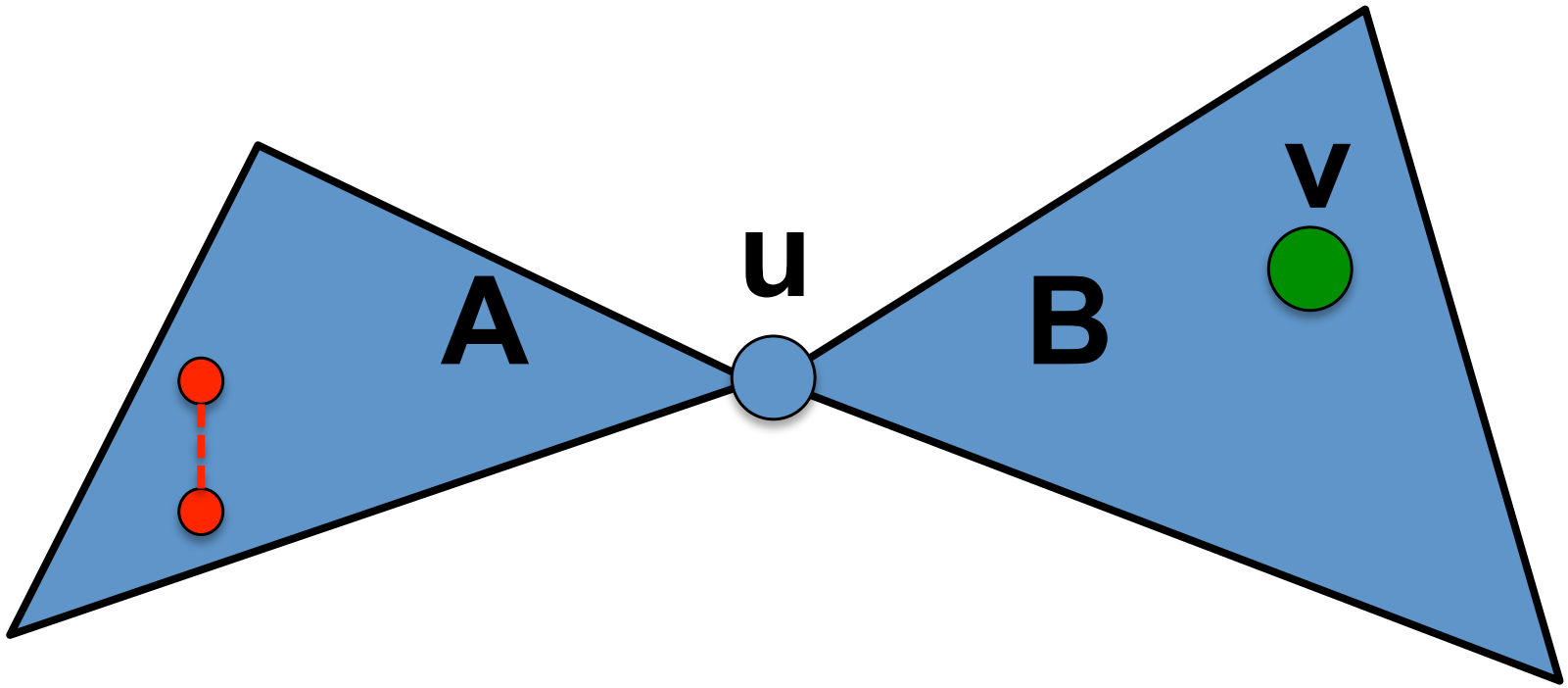
- Usual cases: $dd = 0$, $dd = 1$, $dd > 1$

Pruning - level difference

Algorithm 2: Simple work filtering

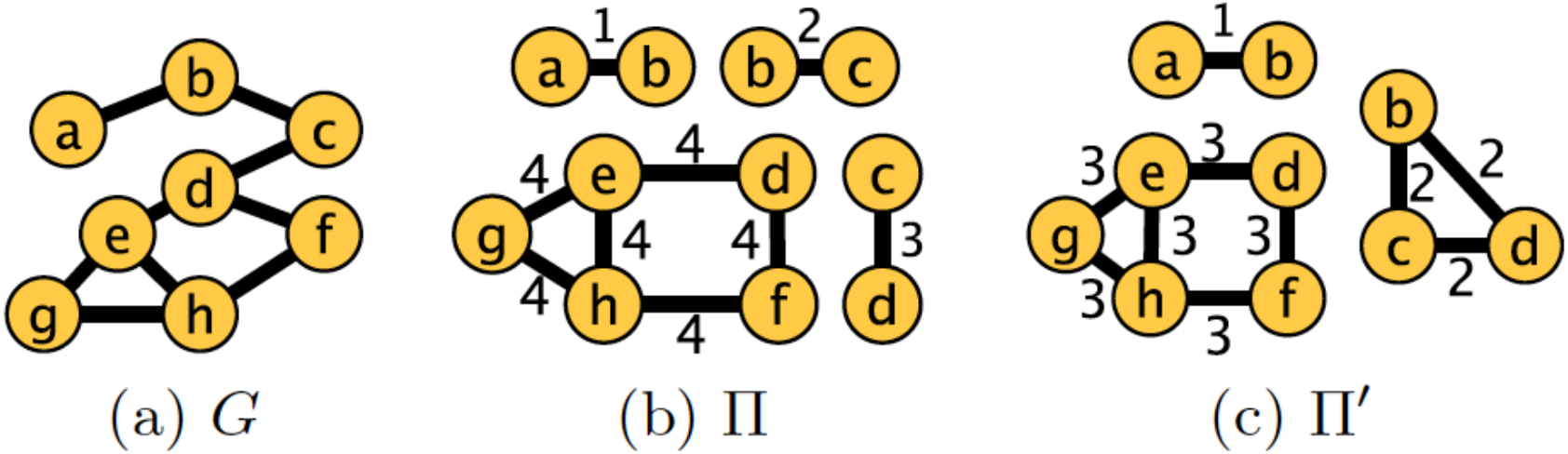
Data: $G = (V, E)$, $\texttt{cc}[.]$, uv
Output: $\texttt{cc}'[.]$
$G' \leftarrow (V, E \cup \{uv\})$
$du[.] \leftarrow$ SSSP(G, u) ▷ distances from u in G
$dv[.] \leftarrow$ SSSP(G, v) ▷ distances from v in G
for each $s \in V$ **do**
  if $|du[s] - dv[s]| \leq 1$ **then**
    $\texttt{cc}'[s] = \texttt{cc}[s]$
  else
    ▷ use the computation in Algorithm 1 with G'
return $\texttt{cc}'[.]$

Pruning - biconnected components



- If graph has articulation points
- Change in A can change closeness of any vertex in B
- It is enough to compute change for u (constant factor is added for the rest of B)

Maintaining biconnected decomposition



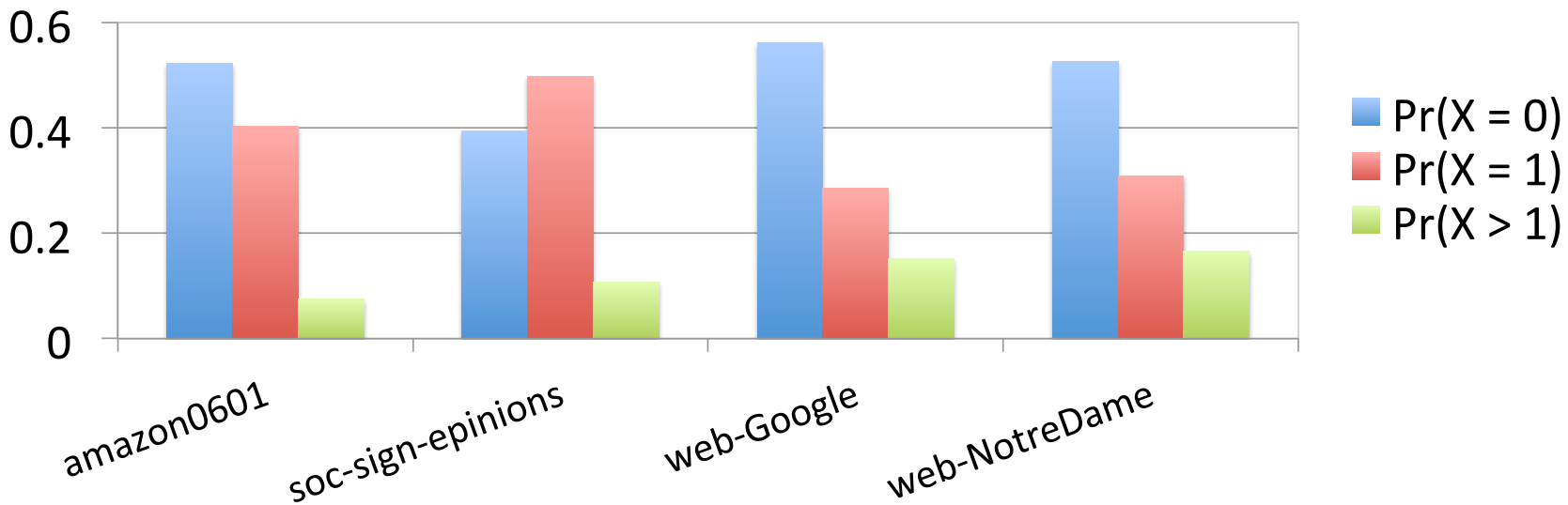
(a) G (b) Π (c) Π'

- Assume edge (b, d) added
- Similar to QUBE

SSSP hybridization

- BFS can be performed in two ways
- Top-down: process vertices at distance d to find vertices at distance $d + 1$
- Bottom-up: after vertices at distance d are found, process all unprocessed vertices to see if they are neighbors of the frontier
- Top-down is better for initial rounds, bottom-up better for final rounds
- Hybridization: use best option at each round

Fraction of cases



- Probability distribution for level difference *dd*
- Most edges are easy cases

Speedup

Graph	Time (secs)					Speedups				Filter
	CC	CC-B	CC-BL	CC-BLI	CC-BLIH	CC-B	CC-BL	CC-BLI	CC-BLIH	time (secs)
hep-th	1.413	0.317	0.057	0.053	0.048	4.5	24.8	26.6	29.4	0.001
PGPgiantcompo	4.960	0.431	0.059	0.055	0.045	11.5	84.1	89.9	111.2	0.001
astro-ph	14.567	9.431	0.809	0.645	0.359	1.5	18.0	22.6	40.5	0.004
cond-mat-2005	77.903	39.049	5.618	4.687	2.865	2.0	13.9	16.6	27.2	0.010
Geometric mean	9.444	2.663	0.352	0.306	0.217	3.5	26.8	30.7	43.5	0.003
soc-sign-epinions	778.870	257.410	20.603	19.935	6.254	3.0	37.8	39.1	124.5	0.041
loc-gowalla	2,267.187	1,270.820	132.955	135.015	53.182	1.8	17.1	16.8	42.6	0.063
web-NotreDame	2,845.367	579.821	118.861	83.817	53.059	4.9	23.9	33.9	53.6	0.050
amazon0601	14,903.080	11,953.680	540.092	551.867	298.095	1.2	27.6	27.0	50.0	0.158
web-Google	65,306.600	22,034.460	2,457.660	1,701.249	824.417	3.0	26.6	38.4	79.2	0.267
wiki-Talk	175,450.720	25,701.710	2,513.041	2,123.096	922.828	6.8	69.8	82.6	190.1	0.491
DBLP-coauthor	115,919.518	18,501.147	288.269	251.557	252.647	6.2	402.1	460.8	458.8	0.530
Geometric mean	13,884.152	4,218.031	315.777	273.036	139.170	3.2	43.9	50.8	99.7	0.146

- Speedup of 2 orders of magnitude
- Mostly due to level pruning
- Biconnected decomposition and hybridization also give good speedups

Scalable Online Betweenness Centrality in Evolving Graphs

N. Kourtellis, G. De-Francisci-Morales, F. Bonchi

TKDE: IEEE Transactions on Knowledge and Data Engineering (2015)

Intuition

- Incremental, exact, space-efficient, out-of-core, parallel version of Brandes'
- Handles edge addition and removal
- Vertex and edge betweenness
- Scalable to graphs with millions of vertices

Algorithm

- Run a modified Brandes' on the initial graph
- Keep track of d, σ, δ in a $\mathbb{S}$DAG (no P)
- On edge update, adjust the $\mathbb{S}$DAG and update b

Input: Graph $G(V, E)$ and edge update stream E_S
Output: $VBC'[V']$ and $EBC'[E']$ for updated $G'(V', E')$

Step 1: Execute Brandes' alg. on G to create & store data structures for incremental betweenness.

Step 2: **For each** update $e \in E_S$, execute Algorithm 1.

Step 2.1 Update vertex and edge betweenness.

Step 2.2 Update data structures in memory or disk for next edge addition or removal.

Data structure

- $\mathbb{S}\text{DAG}_s$ for each source $s \in V$
- $\mathbb{S}\text{DAG}$ contains d, σ, δ for each other vertex $t \in V$
- No predecessors P, re-scan neighbors and use d to find them
  - Save memory - space complexity $O(n^2)$
  - Fixed size data structure - efficient out-of-core management
  - Same time complexity $O(nm)$ - in practice, makes the algorithm faster

Pivot

- When adding or removing an edge, consider $dd = |d_{su} - d_{sv}|$
- Three cases: $dd = 0$, $dd = 1$, $dd > 1$ (analogous to Green et al.)
- Last case $dd > 1$ hardest - structural changes in $\mathbb{S}\text{DAG}$
- Find pivots to discover structural changes

Definition (Pivot)

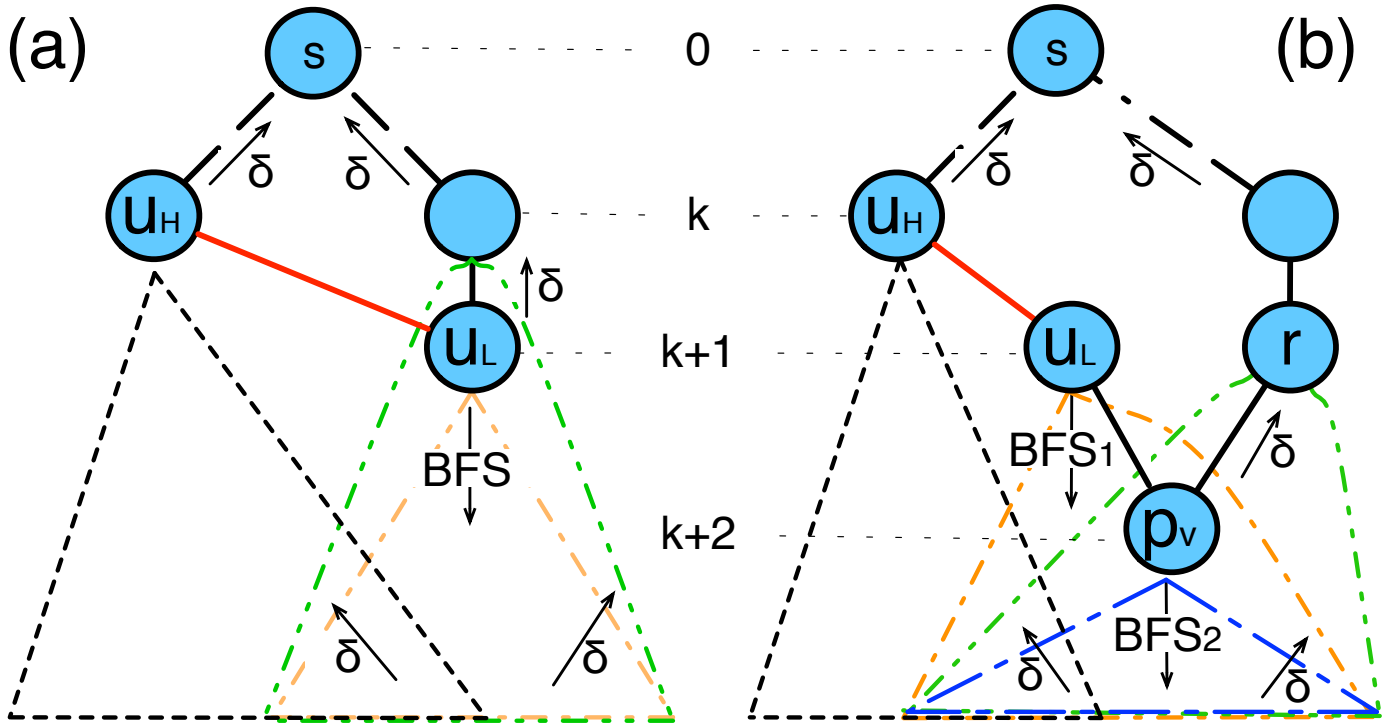
Let s be the current source, let d and d' be the distance before and after an update, respectively, we define pivot a vertex $p \mid d(s,p) = d'(s,p) \wedge \exists\, w \in \Gamma(p)\colon d(s,w) \neq d'(s,w)$.

- Pivots' distance unchanged $\rightarrow$ use as starting points to correct distances

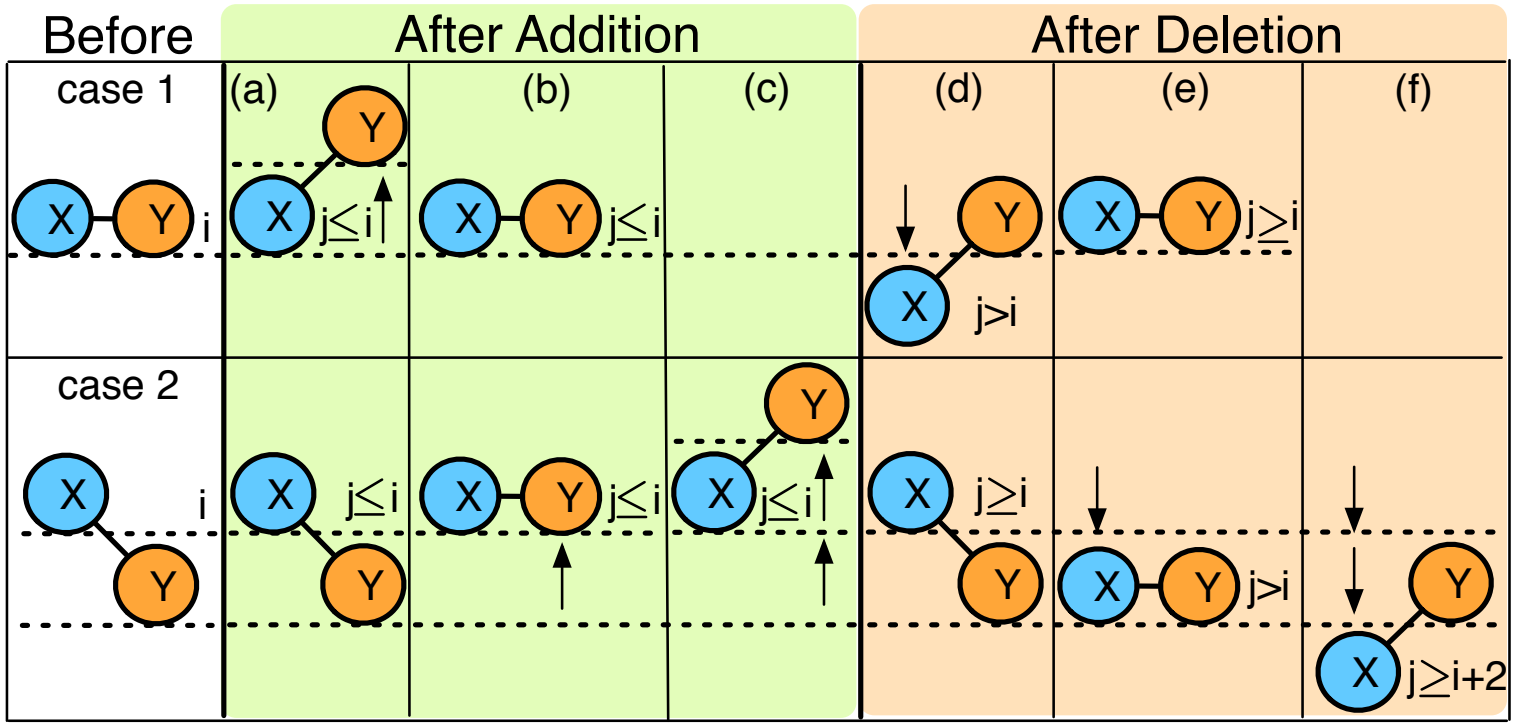
Finding pivots

- Addition - pivots in sub-dag rooted in $u_L = v$
- vertices moved closer must be reachable from u_L
- Can be found during exploration while fixing σ

- Removal - pivots may be anywhere
- Need one exploration to find them
- Need separate exploration from found pivots to correct distances



Structural changes

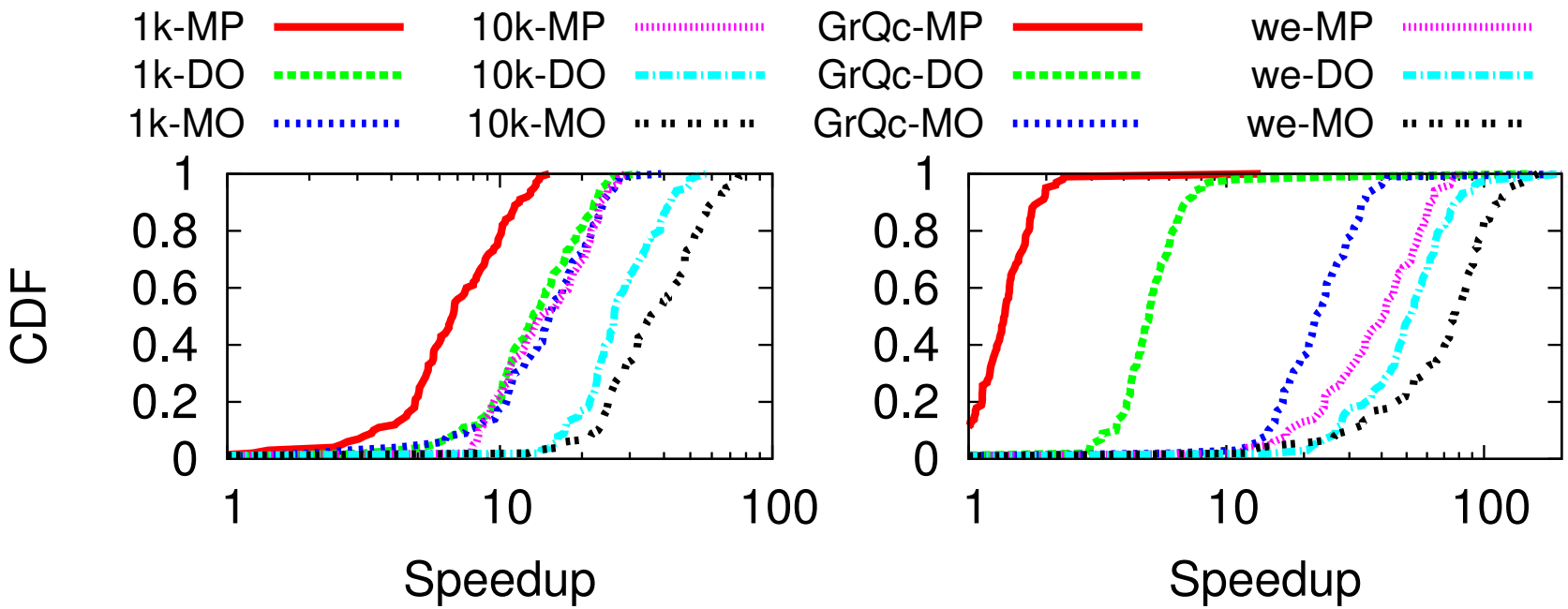


- Consider $x \in \Gamma(y)$, x can either be a sibling or a predecessor of y
- Each case requires slightly different combination of corrections for d, σ, δ
- y is pivot in 1d, 2e, 2f
- Removal for case 1d can be optimized (pivot y is sibling of x)

Scalability

- Out-of-core - stream $\mathbb{S}_{\mathrm{DAG}}$ from disk
  - In-place update on disk to minimize writes
- Columnar storage for d, σ, δ
  - Read only d, skip rest if $dd = 0$
- Parallelization - coarse grained over s
  - Implementation in MapReduce
  - Amenable to Apache Storm/Flink/Spark

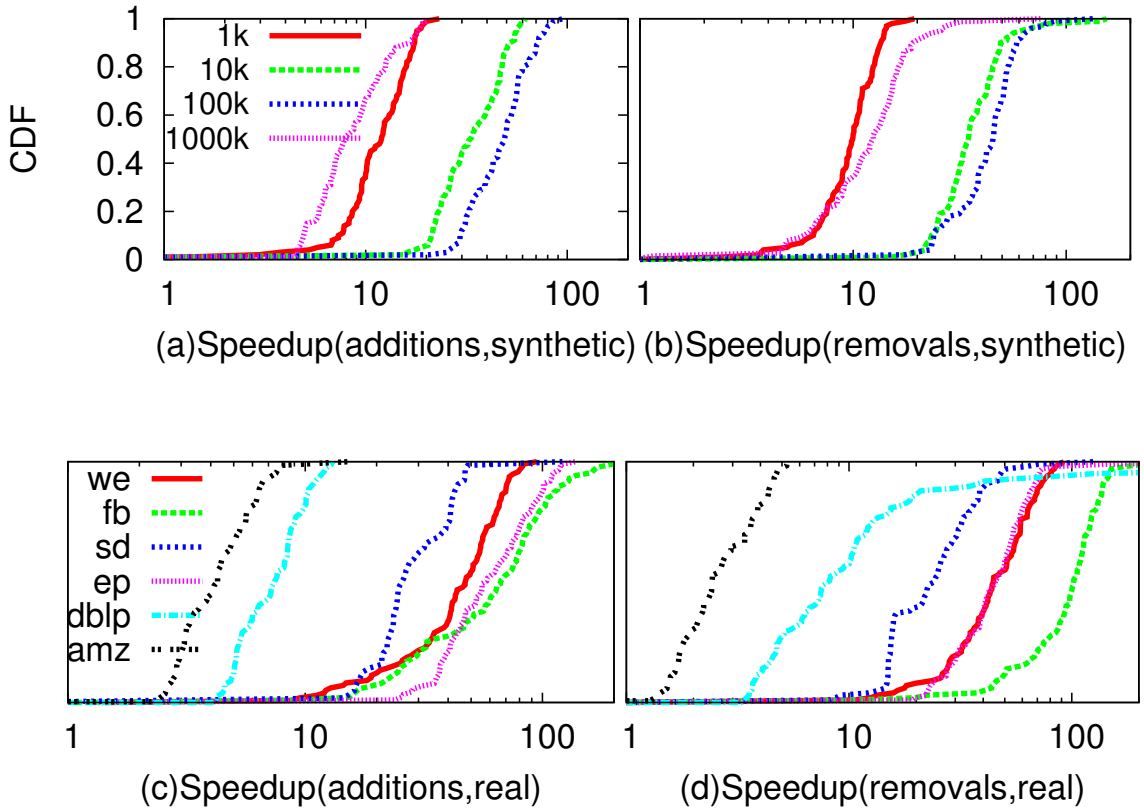
Results



Speedup over Brandes' on synthetic and real graphs ($n = 10k$)

- In-memory (M-) version faster than out-of-core (D-)
- Without predecessor (-O) always faster than with predecessors (-P)

Results



Speedup over Brandes' for out-of-core version on synthetic and real graphs ($n = 1M$)

- Out-of-core version scales up to 1M vertices
- Speedup up to 2 orders of magnitude

Conclusions

- Fully dynamic (addition and removal)
- Algorithm can scale to graphs with realistic size
- Ideal horizontal scalability
- $O(n^2)$ space bottleneck

Approximation Algorithms

Why should we look for an approximation?

Static graphs

- many interesting networks are web-scale;
- computing the exact centralities can be extremely expensive;
- is there a real reason (i.e., application) to require the exact values?

Dynamic graphs

- exact centralities change at all times;
- not worth chasing for highly volatile quantities;

In both cases, high quality approximations are sufficient in practice

What kind of approximation

- v: vertex with exact centrality $c(v)$
- $\tilde{c}(v)$: value that "approximates" $c(v)$

Definition (Absolute error)

$$\mathrm{err}_{\mathrm{abs}}(v) = |c(v) - \tilde{c}(v)|$$

Definition (Relative error)

$$\mathrm{err}_{\mathrm{rel}}(v) = |c(v) - \tilde{c}(v)|/c(v)$$

Definition ((ε, δ)-approximation)

- Let $\varepsilon \in (0, 1)$ and $\delta \in (0, 1)$;
- a (ε, δ)-approximation is a set $\{\tilde{c}(v), v \in V\}$ of n values, such that
$$\Pr\left(\exists v \in V \text{ s.t. } \mathrm{err}(v) > \varepsilon\right) \leq \delta;$$
- it offers uniform probabilistic guarantees over all the nodes;
- it assumes normalized versions of centrality (i.e., in $[0, 1]$).

Sampling

Many of the algorithms we present are sampling-based.

General Sampling Based Algorithm

1. Select independently at random (not all uniformly) a small set of objects (e.g., single vertices, pair of vertices, shortest paths);
2. Perform some computation using these objects (e.g., SSSP from vertex);
3. Use the results of the computation to estimate the centrality of all nodes;

Sampling

Why sampling?

By only select a small subset of the "objects" (instead of the whole set), computing the approximation is faster than computing the exact values

Questions for sampling algorithms

- What "objects" to sample?
- How to sample?

  If sampling procedure is slow, then the advantages are lost;
- How many objects to sample in order to guarantee an (ε, δ)-approximation?

Outline

- Approximation algorithms for static graphs
  - A sampling-based algorithm for closeness
  - A sampling+pivoting algorithm for closeness
  - Two sampling-based algorithms for betweenness
- Approximation algorithms for dynamic graphs
  - Two sampling-based algorithms for betweenness

Approximation Algorithms for Static Graphs

Fast approximation of centrality

D. Eppstein, J. Wang

Journal of Graph Algorithms and Applications (2004)

Idea

Interested in approximating closeness:

$$c(x) = \frac{n-1}{\sum_{y \neq x} d(x,y)}$$

(inverse of the average distance)

Fastest-known exact algorithm: APSP

I.e., run Dijkstra's algorithm from each vertex v

Idea: only run Dijkstra from a few sources!

Warning

The algorithm actually computes an approximation for the inverse of closeness:

$$c^{-1}(v) = \frac{\sum_{y \neq x} d(u,v)}{n-1}$$

(effectively the average distance)

Algorithm

- Let k be the number of sources to obtain the desired approximation;
- For $i = 1, \ldots, k$:
  - pick a vertex u_i uniformly at random
  - run Dijkstra from u_i
- Let

$$\widetilde{c^{-1}}(v) = \frac{n}{n-1} \frac{\sum_{i=1}^{k} d(u_i, v_i)}{k}$$

Theorem

$\mathbb{E}\left[\widetilde{c^{-1}}(v)\right] = c^{-1}(v)$.

Question

How large should k be to get a good approximation of c^{-1}?

How much to sample

Lemma

Let Δ be the diameter of the graph and let $\varepsilon, \delta \in (0,1)$. If

$$k \geq \frac{2}{\varepsilon^2}\left(\ln 2 + \ln n + \ln \frac{1}{\delta}\right)$$

Then, with probability at least $1-\delta$

$$\left|\widetilde{c^{-1}}(v) - c^{-1}(v)\right| \leq \Delta\varepsilon, \text{ for all } v \in V$$

Proof

1. Hoeffding inequality to bound the error of a single vertex;
2. Union bound to get uniform guarantees.

Running time: $O\left(\frac{\log n - \log \delta}{\varepsilon^2}(n \log n + m)\right)$.

Computing Classic Closeness Centrality, at Scale

E. Cohen, D. Delling, T. Pajor, R. F. Werneck

COSN '14: ACM Conference on Social Networks (2014)

Issues with sampling

- Assume that the distance distribution from a vertex v has a heavy tail, then the average distance

$$\mathsf{c}^{-1}(v) = \frac{\sum_{u \neq v} d(u, v)}{n - 1}$$

  is dominated by few distant vertices;
- it is unlikely that these vertices are among the k that are sampled
- Hence the sample average

$$\widetilde{\mathsf{c}^{-1}}(v) = \frac{n}{n-1} \frac{\sum_{i=1}^{k} d(u_i, v_i)}{k}$$

  is a poor estimator of the average distance $\mathsf{c}^{-1}(v)$.
- Sampling along can't give us small relative error

Pivoting

Definition

Pivot The pivot $p(v)$ of a vertex v is the sampled vertex which is closest to v ($p(v) \in \mathcal{S}$).

- We have the exact value of $c^{-1}(p(v))$, can we leverage it?
- The average SP distance $c^{-1}(v)$ of v is "close" to $c^{-1}(p(v))$:

$$c^{-1}(p(v)) - d(v, p(v)) \leq c^{-1}(v) \leq c^{-1}(p(v)) - d(v, p(v))$$

- One can actually prove that, with high probability,

$$c^{-1}(p(v)) + d(v, p(v)) \leq 3c^{-1}(v) + O(1)$$

Pivoting by itself is not satisfactory: the relative error is still somewhat large.

Idea: combine sampling and pivoting into a hybrid estimator

Hybrid Estimator

For each vertex v with pivot $p(v)$, split the set $V \setminus \mathcal{S}$ into three sets:

- $L(v)$: vertices in $V \setminus \mathcal{S}$ at distance at most $d(v, p(v))$ from $p(v)$;
- $HC(v)$: vertices in $\mathcal{S}$ with distance greater than $d(v, p(v))$ from $p(v)$.
- $H(v)$: vertices in $V \setminus \mathcal{S}$ at distance greater than $d(v, p(v))$ from $p(v)$.

The hybrid estimator is

$$\widetilde{\mathrm{c}^{-1}}(v) = \frac{1}{n-1}\left(\sum_{u \in H(v)} d(p(v), u) + \sum_{u \in HC(v)} d(u, v) + \frac{|L(v)|}{|L(v) \cap \mathcal{S}|} \sum_{u \in L(v) \cap \mathcal{S}} d(u, v)\right)$$

We have $\mathbb{E}[\widetilde{\mathrm{c}^{-1}}(v)] \neq \mathrm{c}^{-1}(v)$.

Guarantees

Theorem

- *With* $k = 1/\varepsilon^3$, *the hybrid estimator has normalized RMSE* $O(\varepsilon)$.
- *With* $k = \varepsilon^{-3} \ln n$, *the maximum relative error is* $O(\varepsilon)$ *w.h.p.*

Experiments

				Exact	**Sampling**		**Pivoting**		**Hyb.-0.1**		**Hyb.-ad**	
type	instance	$\lvert V\rvert$ $[\cdot 10^3]$	$\lvert E\rvert$ $[\cdot 10^3]$	time $\approx$ [h:m]	err. [%]	time [sec]	err. [%]	time [sec]	err. [%]	time [sec]	err. [%]	time [sec]
road	fla-t	1 070	1 344	59:30	5.4	24.4	3.2	21.6	2.5	28.3	2.8	73.2
	usa-t	23 947	28 854	44 222:06	2.9	849.4	3.7	736.4	2.0	2 344.3	2.6	9 937.9
grid	grid20	1 049	2 095	70:34	4.3	26.5	3.5	26.8	2.9	29.2	3.3	69.7
triang	buddha	544	1 631	19:07	3.6	14.5	3.3	13.6	2.4	15.9	3.2	30.7
	buddha-w	544	1 631	21:25	3.5	16.4	2.6	15.5	2.2	18.5	2.9	38.1
	del20-w	1 049	3 146	72:06	2.7	27.4	3.6	26.7	2.6	32.6	2.7	71.0
	del20	1 049	3 146	67:54	4.1	25.6	5.3	25.2	3.7	27.0	3.6	54.7
game	FrozenSea	753	2 882	38:25	3.0	22.1	4.1	20.2	2.1	24.0	3.4	49.3

The hybrid estimator is better than just-sampling and just-pivoting.

Summary for closeness

- Sampling can help, but not alone
- Pivoting alone is not good
- The hybrid approach is promising, but the sample size results are somewhat disappointing (very large sample sizes!)

More work to do!

Centrality Estimation in Large Networks

U. Brandes, C. Pich

International Journal of Bifurcation and Chaos (2007)

Betweenness centrality

We consider a normalized version:

$$b(v) = \frac{1}{n(n-1)} \sum_{s,t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} \in [0,1]$$

- σ_{st}: number of SPs from s to t
- $\sigma_{st}(v)$: number of SPs from s to t going through v

Exact algorithm: Brandes' Algorithm

1. Run Dijkstra's algorithm from each source vertex s
2. After each run, perform aggregation by walking SP DAG backwards

Idea: run Dijkstra only from a few sources (as in EW'01)

How can one get an (ε, δ)-approximation?

$k \leftarrow \frac{1}{\varepsilon^2}\left(\ln n + \ln 2 + \ln \frac{1}{\delta}\right)$ // sample size
$\tilde{\mathrm{b}}(v) \leftarrow 0$, for all $v \in V$
for $i \leftarrow 1, \ldots, k$ **do** // Brandes' algo iterates over V
  $v_i \leftarrow$ random vertex from V, chosen uniformly
  Perform single-source SP computation from v_i
  Perform partial aggregation, updating $\tilde{\mathrm{b}}(u)$, $u \in V$, like in exact algorithm
end
Output $\{\tilde{\mathrm{b}}(v), v \in V\}$

Theorem

The output is a (ε, δ)-approximation:

$$\Pr\left(\exists v \in V \text{ s.t. } |\tilde{\mathrm{b}}(v) - \mathrm{b}v| > \varepsilon\right) \leq \delta$$

How do they prove it?

Start with bounding the deviation for a single vertex v (Hoeffding inequality):

$$\Pr(|\tilde{b}(v) - b(v)| > \varepsilon) \leq 2e^{-2k\varepsilon^2}$$

Then take the union bound over n vertices to ensure uniform convergence

The sample size k must be such that

$$2e^{-2k\varepsilon^2} \leq \frac{\delta}{n}$$

That is, to get an (ε, δ)-approximation, we need

$$k \geq \frac{1}{2\varepsilon^2}\left(\ln n + \ln 2 + \ln \frac{1}{\delta}\right)$$

Better Approximation of Betweenness Centrality

R. Geisberger, P. Sanders, D. Schultes

ALENEX (2008)

Issues with standard estimator

The standard estimator

$$\tilde{b}(v) = \frac{1}{k} \sum_{i=1}^{k} \delta_{u_i}(v)$$

produces large overestimates for unimportant vertices close to a sampled vertex

Example

- Let v be a degree-two vertex connecting a degree-one vertex u to the rest of the network;
- If u is sampled, then $\tilde{b}(v)$ overestimates $b(v)$ by a factor of n/k

Possible solution: stop vertices from "profiting" for being near a sampled vertex.

A new sampling scheme

Idea: sample pairs (s, d) of vertex and direction ("forward" or "backward")

- When sampling $(s, \text{forward})$
  - run Dijkstra from s
- When sampling $(t, \text{backward})$
  - virtually flip direction of edges (if directed graph);
  - run Dijkstra from s

We need to adapt the estimator $\tilde{\mathrm{b}}(v)$.

New estimator

For a vertex v, define

$$g_v(u,d) = \begin{cases} \sum_{t \in V, t \neq u,v} \frac{\sigma_{ut}(v)}{\sigma_{ut}} \frac{d(u,v)}{d(v,t)} & \text{if } d = \text{forward} \\ \sum_{t \in V, t \neq u,v} \frac{\sigma_{ut}(v)}{\sigma_{ut}} \left(1 - \frac{d(u,v)}{d(v,t)}\right) & \text{if } d = \text{backward} \end{cases}$$

The new estimator for $\mathrm{b}(v)$ is

$$\tilde{\mathrm{b}}(v) = \frac{2}{k} \sum_{i=1}^{k} g_v(u_i, d_i)$$

The factor 2 corrects for the reduced sampling probabilities $(1/2n)$

Theorem

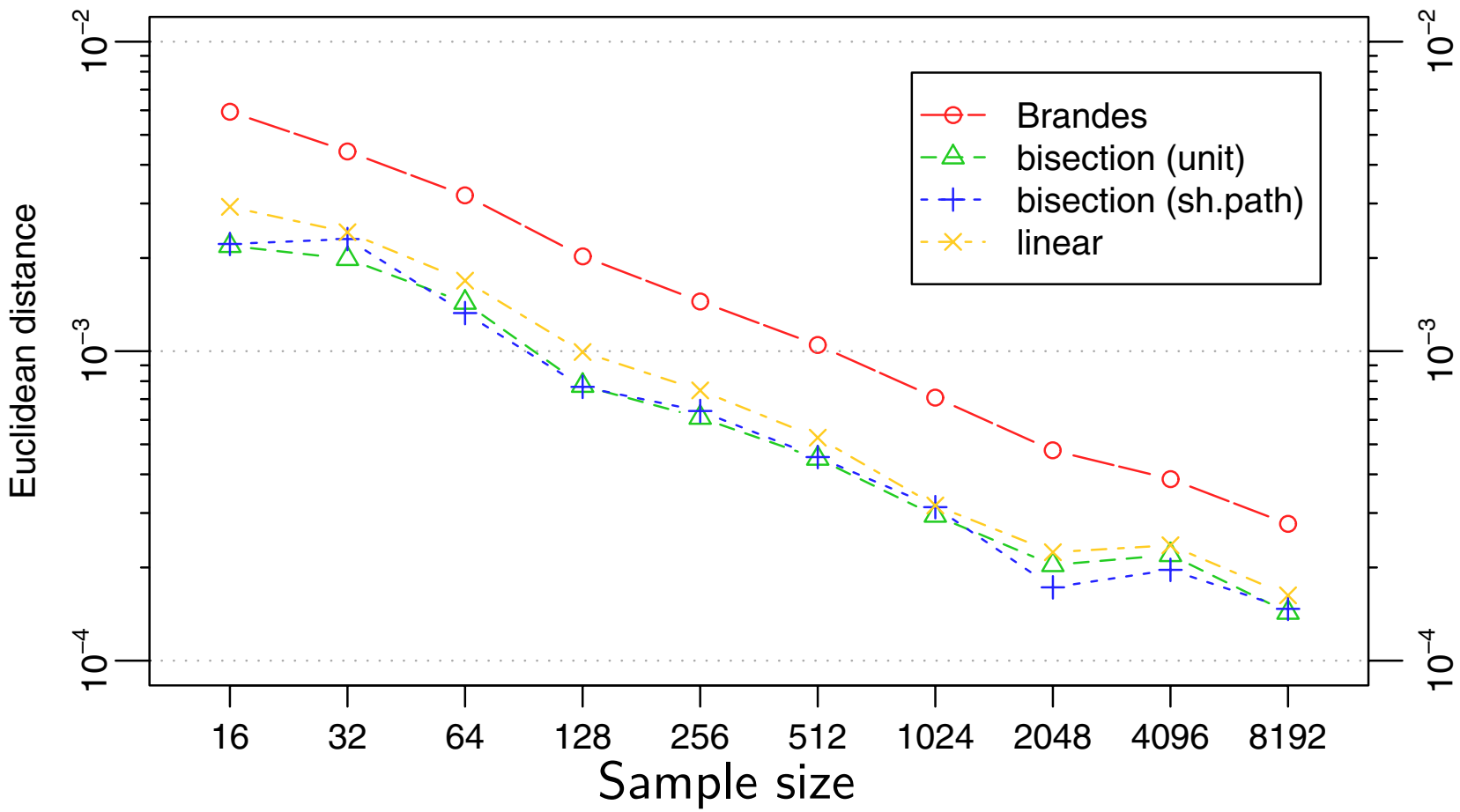
If

$$k \geq \frac{1}{2\varepsilon^2} \left(\ln 2 + \ln n + \ln \frac{1}{\delta}) \right),$$

then the output is a (ε, δ)-approximation:

$$\Pr\left(\exists v \in V \text{ s.t. } |\tilde{\mathrm{b}}(v) - \mathrm{b}v| > \varepsilon\right) \leq \delta$$

Experiments



Euclidean distance between the vector of exact centralities and the vector of estimated centralities.

Fast Approximation of Betweenness Centrality through Sampling

M. Riondato, E. M. Kornaropoulos

DMKD: Data Mining and Knowledge Discovery (2015)

What is wrong with this sampling approach?

1) The algorithm needs

$$k \geq \frac{1}{2\varepsilon^2}\left(\ln n + \ln 2 + \ln \frac{1}{\delta}\right)$$

- This is loose due to the union bound, and does not scale well (experiments)
- The sample size depends on $\ln n$. This is not the right quantity: not all graphs of n nodes are equally "difficult": e.g., the n-star is "easier" than a random graph

The sample size k should depend on a more specific characteristic quantity of the graph

2) At each iteration, the algorithm performs a SSSP computation
Full exploration of the graph, no locality

How can we improve the sample size?

[R. and Kornaropoulos, 2015] present an algorithm that:

1) uses a sample size which depends on the vertex-diameter, a characteristic quantity of the graph.

The derivation uses the VC-dimension of the problem;

2) samples SPs according to a specific, non-uniform distribution over the set $\mathbb{S}_G$ of all SPs in the graph. For each sample, it performs a single $s - t$ SP computation

- More locality: fewer edges touched than single-source SP
- Can use bidirectional search / A*, ...

What is the algorithm?

$\mathrm{VD}(G) \leftarrow$ vertex-diameter of G `// stay tuned!`

$k \leftarrow \frac{1}{2\varepsilon^2}\left(\lfloor \log_2(\mathrm{VD}(G) - 2) \rfloor + 1 + \ln(1/\delta)\right)$ `// sample size`

$\tilde{b}(v) \leftarrow 0$, for all $v \in V$

for $i \leftarrow 1 \ldots, k$ **do**

  $(u, v) \leftarrow$ random pair of different vertices, chosen uniformly

  $\mathcal{S}_{uv} \leftarrow$ all SPs from u to v `// Dijkstra, trunc. BFS, ...`

  $p \leftarrow$ random element of $\mathcal{S}_{uv}$, chosen uniformly `// not uniform over` $\mathbb{S}_G$

  $\tilde{b}(w) \leftarrow \tilde{b}(w) + 1/k$, for all $w \in \mathrm{Int}(p)$ `// update only nodes along` p

end

Output $\{\tilde{b}(v), v \in V\}$

Theorem

The output $\{\tilde{b}(v), v \in V\}$ is an (ε, δ)-approximation.

VC-dimension

- The Vapnik-Chervonkenkis (VC) dimension is a combinatorial quantity that allows to study the sample complexity of a learning problem;
- It allows to obtain uniform guarantees on sample-based approximations of expectations of all functions in a family $\mathcal{F}$;
- Not easy to compute exactly, somewhat easier to give upper bounds;

Theorem (VC ε-sample)

- *Let $\mathcal{F}$ be a family of functions from a domain $\mathcal{D}$ into $\{0,1\}$;*
- *Let d be an upper bound to the VC-dimension of $\mathcal{F}$;*
- *Let $\varepsilon \in (0,1)$ and $\delta \in (0,1)$*
- *Let $\mathcal{S}$ be a random sample of $\mathcal{D}$ of size*

$$|\mathcal{S}| \geq \frac{1}{\varepsilon^2}\left(d + \ln\frac{1}{\delta}\right)$$

obtained by sampling $\mathcal{D}$ according to a prob. distribution π

- *Then*

$$\Pr\left(\exists f \in \mathcal{F} \text{ s.t. } \left|\frac{1}{|\mathcal{S}|}\sum_{s\in\mathcal{S}} f(s) - \mathbb{E}_\pi[f]\right| > \varepsilon\right) < \delta \ .$$

In other words: if we sample proportionally to the VC-dimension, we can approximate all expectations with their sample averages.

How can we prove the correctness?

We want to prove that the output $\{\tilde{b}(v), v \in V\}$ is an (ε, δ)-approximation

Roadmap:

1. Define betweenness centrality computation as a expectation estimation problem (domain $\mathcal{D}$, family $\mathcal{F}$, distribution π)
2. Show that the algorithm efficiently samples according to π
3. Show how to efficiently compute an upper bound to the VC-dimension
   Bonus: show tightness of bound
4. Apply the VC-dimension sampling theorem

How do we bound the VC-dimension?

Definition (Vertex-diameter)

The vertex-diameter $\mathrm{VD}(G)$ of G is the maximum number of vertices in a SP of G:

$$\mathrm{VD}(G) = \max\{|p|, p \in \mathbb{S}_G\} \ .$$

If G is unweighted, $\mathrm{VD}(G) = \Delta(G) + 1$. Otherwise no relationship
Very small in social networks, even huge ones (shrinking diameter effect)

Computing $\mathrm{VD}(G)$: $\left(2\frac{\text{max. edge weight}}{\text{min. edge weight}}\right)$-approximation via single-source SP

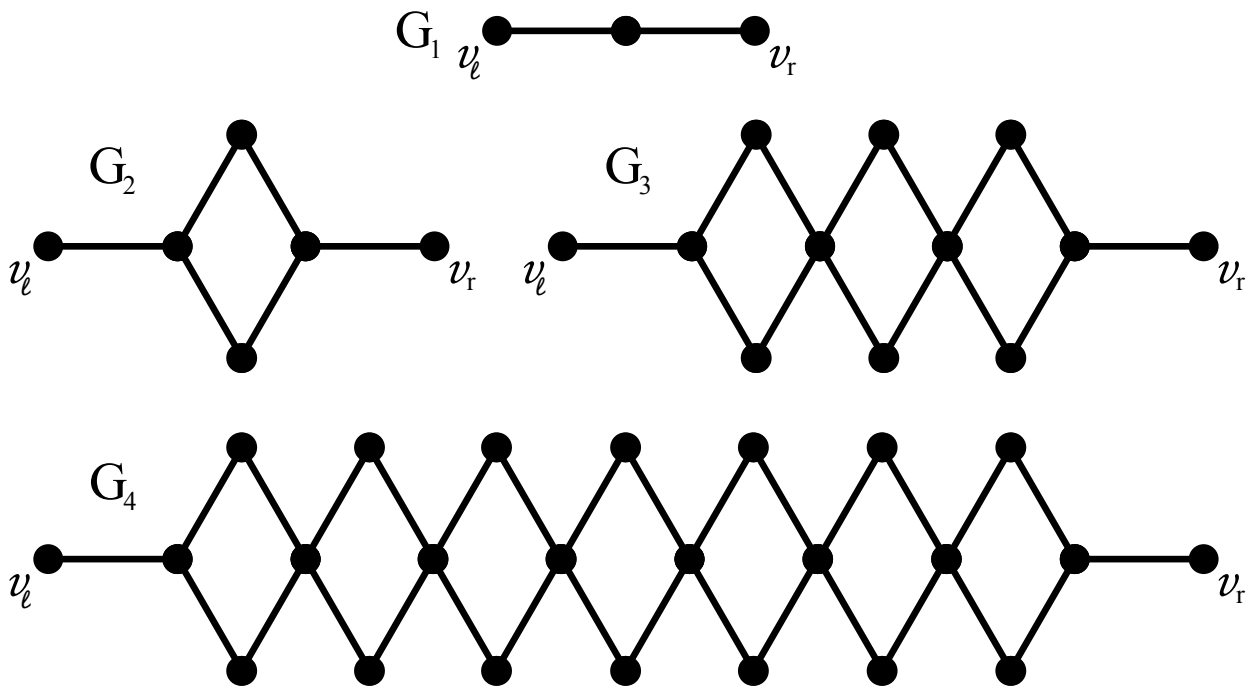
Theorem

The VC-dimension of $(\mathbb{S}_G, F)$ *is at most* $\lfloor \log_2 \mathrm{VD}(G) - 2 \rfloor + 1$

Is the bound to the VC-dimension tight?

Yes! There is a class of graphs with VC-dimension exactly $\lfloor \log_2 \mathrm{VD}(G) - 2 \rfloor + 1$

The Concertina Graph Class $(G_i)_{i \in \mathbb{N}}$:



Theorem

The VC-dimension of $(\mathbb{S}_{G_i}, F)$ *is* $\lfloor \log_2 \mathrm{VD}(G) - 2 \rfloor + 1 = i$

How well does the algorithm perform in practice?

It performs very well!

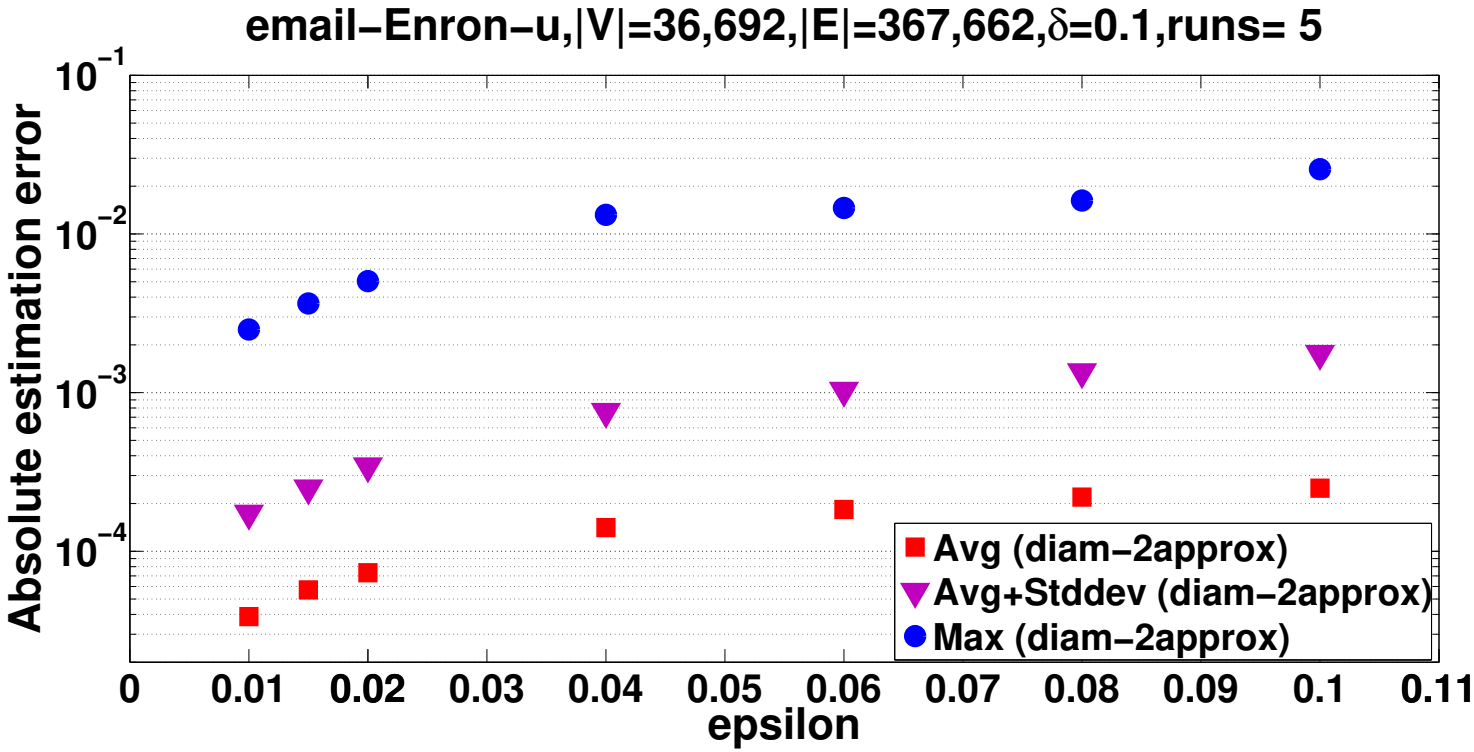
We tested the algorithm on real graphs (SNAP) and on artificial Barabasi-Albert graphs, to evalue its accuracy, speed, and scalability

Results: It blows away the exact algorithm and the union-bound-based sampling algorithm

How accurate is the algorithm?

In $O(10^3)$ runs of the algorithm on different graphs and with different parameters, we always had $|\tilde{b}(v) - b(v)| < \varepsilon$ for all nodes

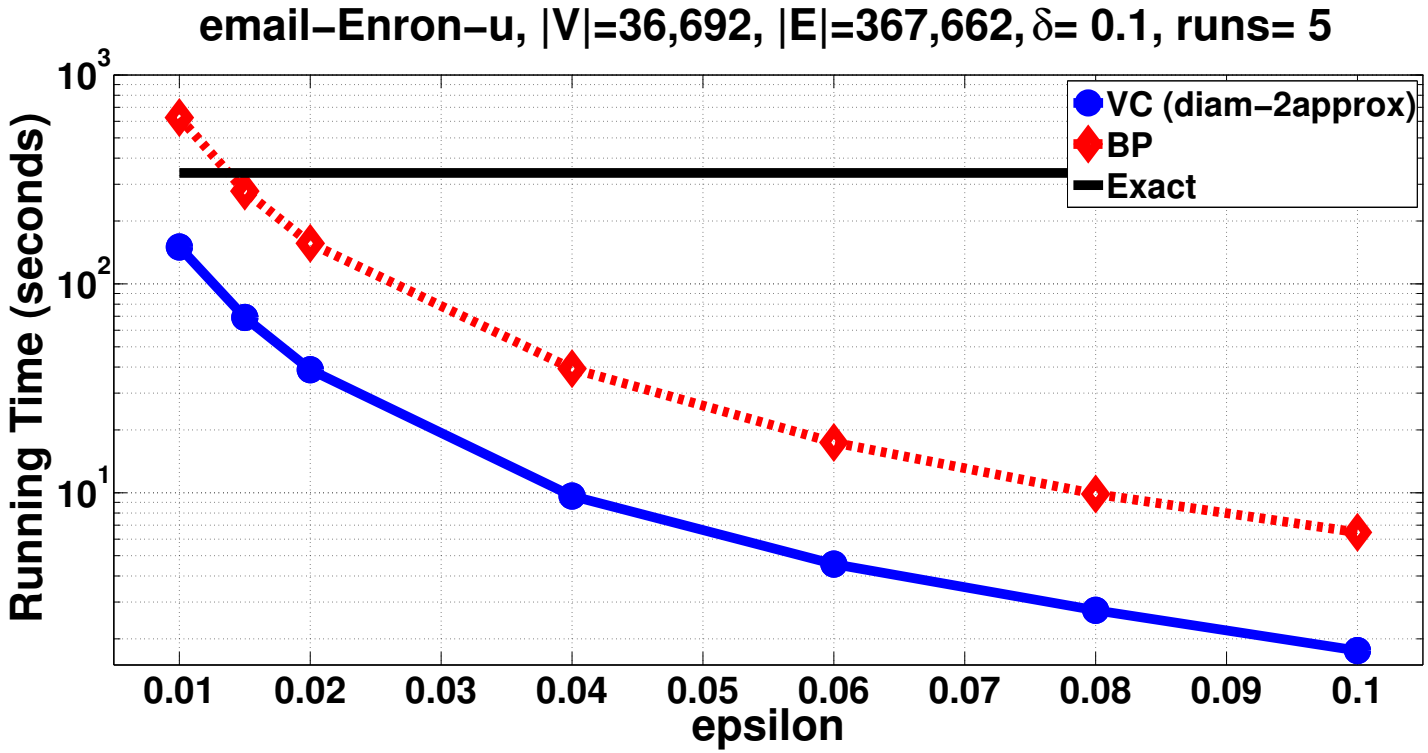
Actually, on average $|\tilde{b}(v) - b(v)| < \varepsilon/8$



How fast is the algorithm?

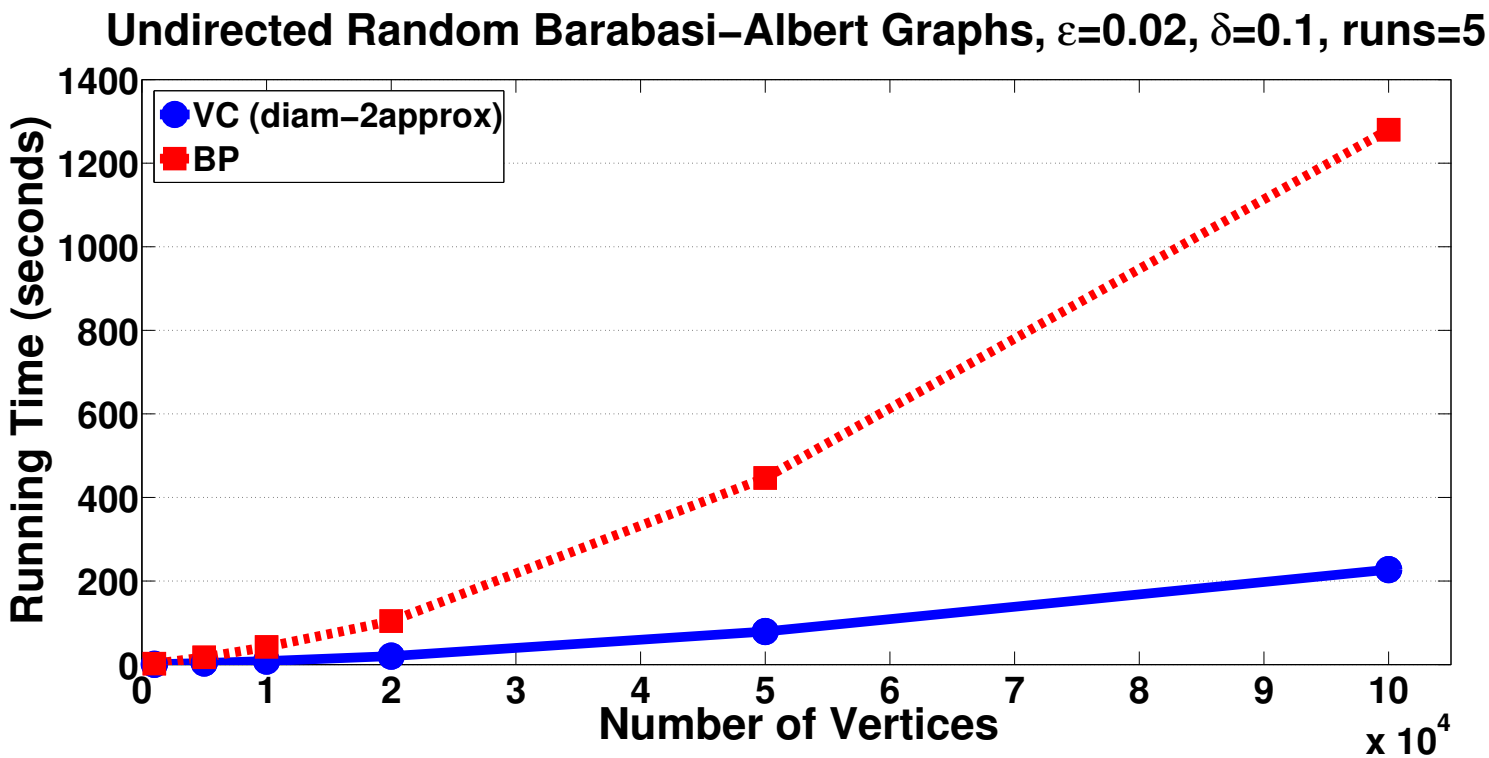
Approximately 8 times faster than the simple sampling algorithm

Variable speedup w.r.t. exact algorithm (200x – 4x), depending on ε



How scalable is the algorithm?

Much more scalable than the simple sampling algorithm, because the sample size does not depend on n



ABRA: Approximating Betweennes Centrality in Static and Dynamic Graphs with Rademacher Averages

M. Riondato, E. Upfal

arXiv (2016)

Issues with RK approach

- For each $s - t$ SP computation, we only use a single SP
  - a lot of wasted work!
- Must compute (upper bound to) the vertex-diameter before we can start sampling
  - Exact computation cannot be done (would be equivalent to obtain exact betweenness)
  - Approximate computation leads to larger-than-necessary sample size

How to solve these issues

- Design a sample scheme that uses all SPs between a sampled pair of vertices
- Use progressive sampling, rather than static sampling
  - Start from small sample size
  - Check stopping condition to verify whether we sampled enough to get a (ε, δ)-approximation
  - If yes, stop, otherwise keep sampling.

How to achieve this: using Rademacher averages (VC-dimension on steroids)

Key ideas

- When backtracking from t to s, follow all SPs, not just one of them, and increase the estimation of all vertices found along the way: no wasted work;
- The stopping condition depends on:
  - the richness of the vectors representing the current estimates of the betweenness of all vertices
  - the current sample size
  - Formulas like this:

$$\frac{1}{1-\alpha}\min_{s\in\mathbb{R}^+}\frac{1}{s}\ln\sum_{\mathbf{v}\in\mathcal{V}_S}\exp(s^2\|\mathbf{v}\|^2/(2\ell^2))+\frac{\ln\frac{2}{\delta}}{2\ell\alpha(1-\alpha)}+\sqrt{\frac{\ln 2/\delta}{2\ell}}$$

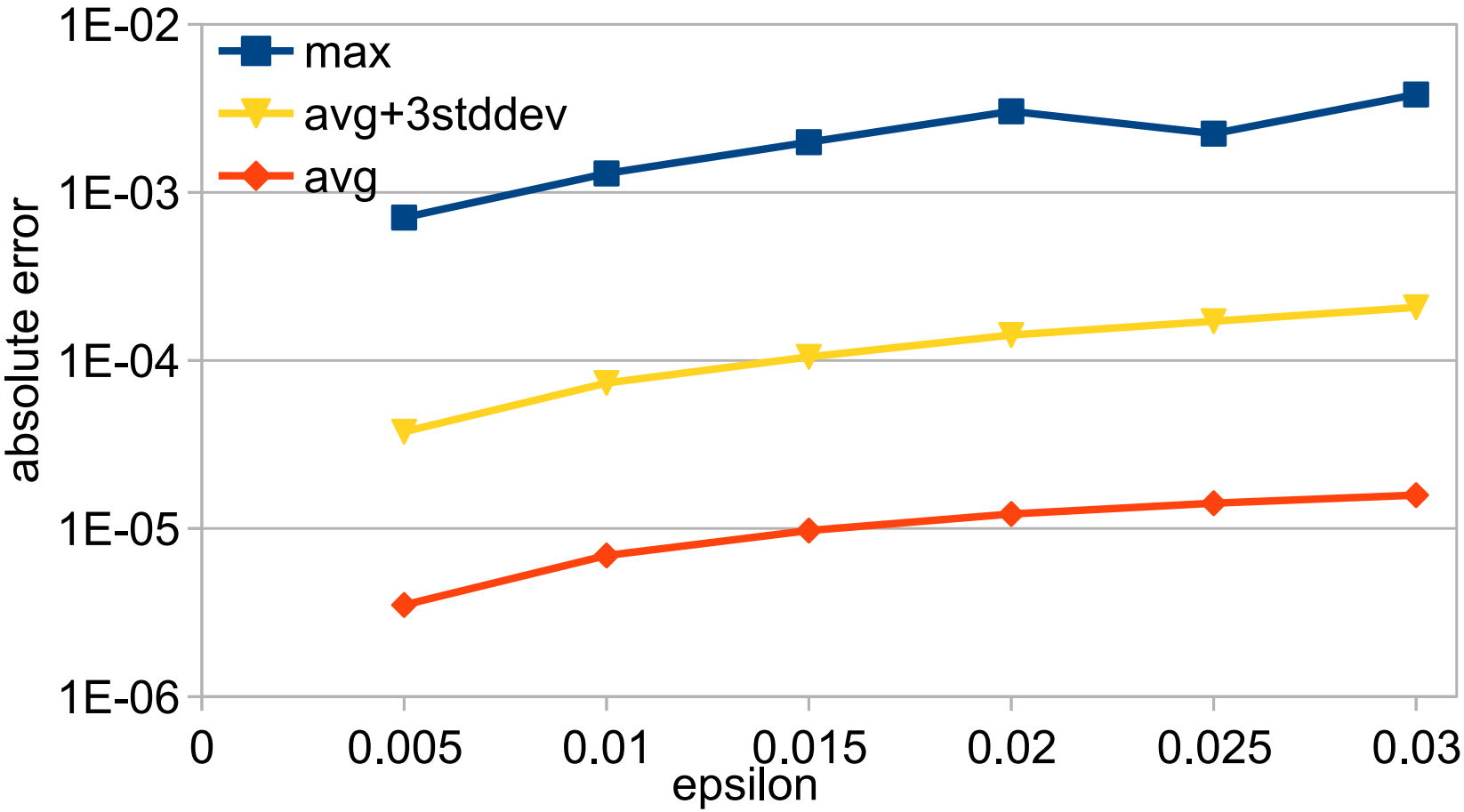
- But it works!

Experiments

Graph	ε	Runtime (sec.)	Speedup w.r.t. BA	Speedup w.r.t. RK	Runtime Breakdown (%) Sampling	Stop Cond.	Other	Sample Size	Reduction w.r.t. RK	Absolute Error ($\times 10^5$) max	avg	stddev
Soc-Epinions1 Directed $\lvert V\rvert = 75,879$ $\lvert E\rvert = 508,837$	0.005	483.06	1.36	2.90	99.983	0.014	0.002	110,705	2.64	70.84	0.35	1.14
	0.010	124.60	5.28	3.31	99.956	0.035	0.009	28,601	2.55	129.60	0.69	2.22
	0.015	57.16	11.50	4.04	99.927	0.054	0.018	13,114	2.47	198.90	0.97	3.17
	0.020	32.90	19.98	5.07	99.895	0.074	0.031	7,614	2.40	303.86	1.22	4.31
	0.025	21.88	30.05	6.27	99.862	0.092	0.046	5,034	2.32	223.63	1.41	5.24
	0.030	16.05	40.95	7.52	99.827	0.111	0.062	3,668	2.21	382.24	1.58	6.37
P2p-Gnutella31 Directed $\lvert V\rvert = 62,586$ $\lvert E\rvert = 147,892$	0.005	100.06	1.78	4.27	99.949	0.041	0.010	81,507	4.07	38.43	0.58	1.60
	0.010	26.05	6.85	4.13	99.861	0.103	0.036	21,315	3.90	65.76	1.15	3.13
	0.015	11.91	14.98	4.03	99.772	0.154	0.074	9,975	3.70	109.10	1.63	4.51
	0.020	7.11	25.09	3.87	99.688	0.191	0.121	5,840	3.55	130.33	2.15	6.12
	0.025	4.84	36.85	3.62	99.607	0.220	0.174	3,905	3.40	171.93	2.52	7.43
	0.030	3.41	52.38	3.66	99.495	0.262	0.243	2,810	3.28	236.36	2.86	8.70
Email-Enron Undirected $\lvert V\rvert = 36,682$ $\lvert E\rvert = 183,831$	0.010	202.43	1.18	1.10	99.984	0.013	0.003	66,882	1.09	145.51	0.48	2.46
	0.015	91.36	2.63	1.09	99.970	0.024	0.006	30,236	1.07	253.06	0.71	3.62
	0.020	53.50	4.48	1.05	99.955	0.035	0.010	17,676	1.03	290.30	0.93	4.83
	0.025	31.99	7.50	1.11	99.932	0.052	0.016	10,589	1.10	548.22	1.21	6.48
	0.030	24.06	9.97	1.03	99.918	0.061	0.021	7,923	1.02	477.32	1.38	7.34
Cit-HepPh Undirected $\lvert V\rvert = 34,546$ $\lvert E\rvert = 421,578$	0.010	215.98	2.36	2.21	99.966	0.030	0.004	32,469	2.25	129.08	1.72	3.40
	0.015	98.27	5.19	2.16	99.938	0.054	0.008	14,747	2.20	226.18	2.49	5.00
	0.020	58.38	8.74	2.05	99.914	0.073	0.013	8,760	2.08	246.14	3.17	6.39
	0.025	37.79	13.50	2.02	99.891	0.091	0.018	5,672	2.06	289.21	3.89	7.97
	0.030	27.13	18.80	1.95	99.869	0.108	0.023	4,076	1.99	359.45	4.45	9.53

- Smaller sample sizes than RK
- Much faster (not just because using smaller sample, also because no need to compute the vertex-diameter)
- Very accurate

Experiments



- More than 10x more accurate than guaranteed, on average;
- More than 100x more accurate than guaranteed, in the best case;
- Close to the guarantee in the worst case: this is good.

Approximation Algorithms for Dynamic Graphs

Fully-Dynamic Approximation of Betweenness Centrality

E. Bergamini, H. Meyerhenke

ESA: European Symposium on Algorithms (2015)

Key ideas

This algorithm builds on:

- the RK sampling-based approximation algorithm;
- existing algorithms to update the SP DAG after an insertion/removal of a batch of edges;

It keeps track of potential modifications to the vertex diameter to understand whether to increase the sample size;

Theorem

After each batch update, the output is an (ε, δ)-approximation.

Updating the DAGs

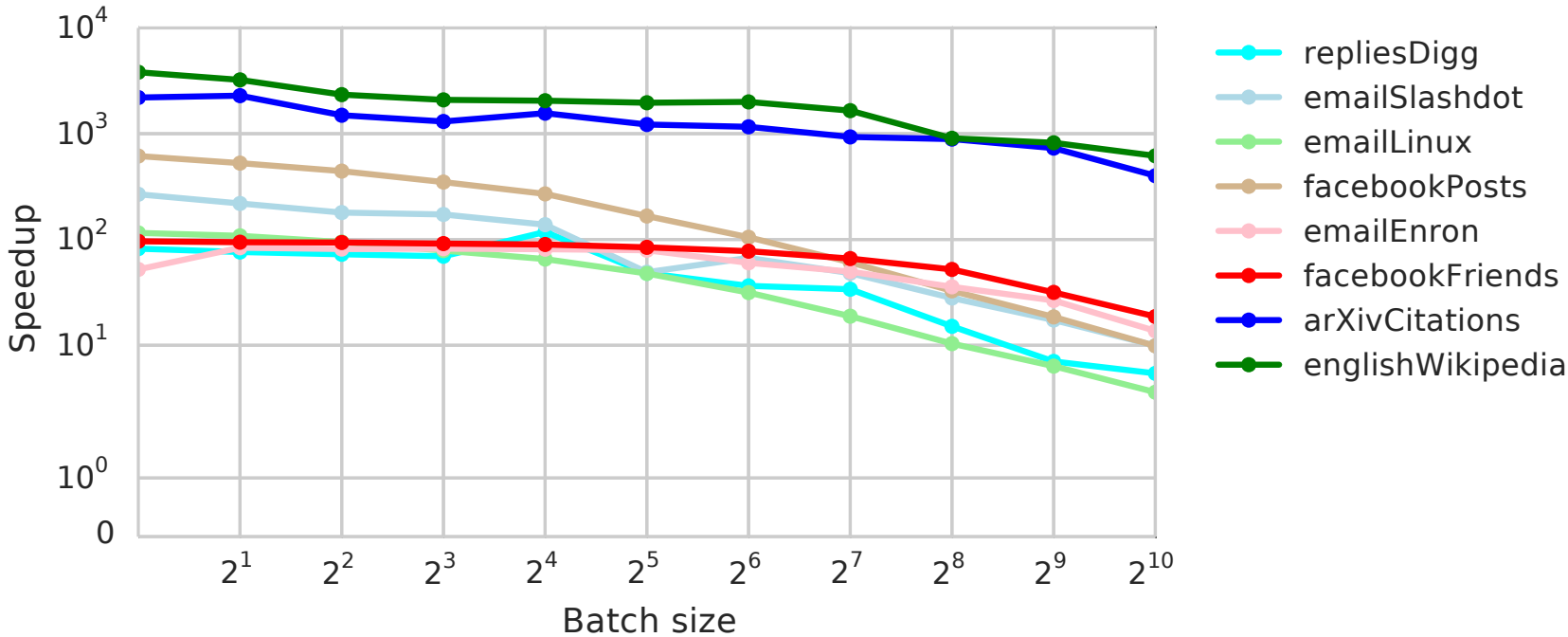
- Never change the set of sampled pairs of vertices, unless a sample was removed or more samples are needed
- What can change is which SP is sampled: if an edge is added, the path we sampled before may no longer be a SP.
- In any case, must save all the SP DAGs between the sampled pair of nodes
- Requires a lot of memory, but is needed in order to be able to update the estimation after the batch update
- The update computation builds on existing algorithms

Keeping track of the vertex diameter

- An edge is removed: the VD may decrease, but no need to change the sample size;
- An edge is added between two existing vertices in the same connected component: no change in the VD, hence no change in sample size
- An edge is added between two existing vertices in two different connected components: the VD may have changed, recomputation is necessary
- An edge is added between an existing vertex and a new vertex: the VD may have increased by one, recomputation is necessary (the model used in this paper does not actually consider the insertion and removal of vertices)

Relying on the vertex diameter is not a great idea, that's why we developed ABRA, the Rademacher Averages-based algorithm.

Experiments



Speedup over RK

Fully Dynamic Betweenness Centrality Maintenance on Massive Networks

T. Hayashi, T. Akiba, Y. Yoshida

VLDB: Very Large Databases (2016)

Key ideas

- Still a sampling-based approximation algorithm, but samples pair of vertices;
- This similar to RU16, but analysis use the union bound, so $O(\varepsilon^{-2} \log n)$ samples, which is a lot;
- Presents a new data structure called hypergraph sketch to keep track of the SP DAGS.
- An additional data structure, called the Two-ball Index, allows to identify the parts of hypergraph sketches that require updates

The Hypergraph Sketch

(effectively a hypergraph)

- For each sampled pair (s, t) of vertices, an hyperedge is added to the hypergraph:

$$e_{st} = \{(v, \sigma_{sv}, \sigma_{v,t}) : v \text{ is on a SP from } s \text{ to } t\}$$

- The estimations $\tilde{b}(v)$ can be obtained from the sketch;
- Handling insertion and removal of edges is straightforward, but must be done efficiently
- Handling insertion and removal of nodes requires to change the set of sampled pair of vertices, i.e., to potentially remove a hyperedge and insert another one;

Vertex Operations

Algorithm 1 Vertex operations

procedure AddVertex(H, v)
  Let G_τ be obtained from $G_{\tau-1}$ by adding v.
  for each $e_{st} \in E(H)$ **do**
    continue with probability $|V_{\tau-1}|^2/|V_\tau|^2$.
    Sample $(s', t') \in (V_\tau \times V_\tau) \setminus (V_{\tau-1} \times V_{\tau-1})$.
    Replace e_{st} by the hyperedge $e_{s't'}$ made from (s', t').

procedure RemoveVertex(H, v)
  Let G_τ be obtained from $G_{\tau-1}$ by deleting v.
  for each $e_{st} \in E(H)$ **do**
    if $s \neq v$ and $t \neq v$ **then continue**.
    Sample $(s', t') \in V_\tau \times V_\tau$ uniformly at random.
    Replace e_{st} by the hyperedge $e_{s't'}$ made from (s', t').

The Two-Ball Index

- For each sampled pair (s, t), maintain a triplet $(\Delta_{st}, \beta^+, \beta^-)$, where
  - $\Delta_{st} = \{d(s, v), v \text{ is on a SP from } s \text{ to } t\}$
  - The ball β^+ is the set of vertices at distance less than some d_s from s, with their distances
  - The ball β^+ is the set of vertices at distance less than some d_t from t, with their distances
- The radiuses of the balls are such that they do not touch and are small.
- The triplets can be built with a bidirectional SP computation from s to t

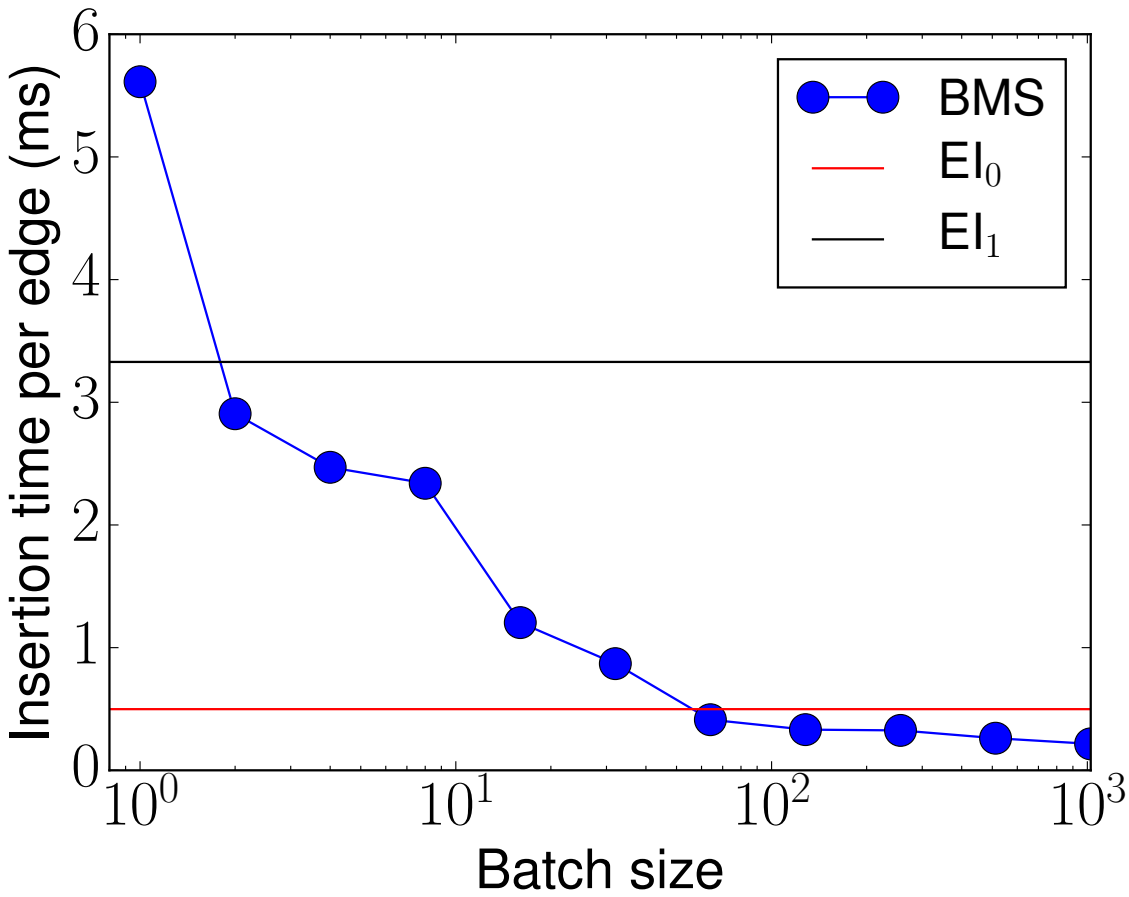
Update Mechanism (for insertion)

Algorithm 2 Update $\overrightarrow{B}(s, d_s)$ after edge (u, v) is inserted

```
1: procedure InsertEdgeIntoBall(u, v, β⃗_s)
2:     Q ← An empty FIFO queue.
3:     if β⃗_s[v] > β⃗_s[u] + 1 then
4:         β⃗_s[v] ← β⃗_s[u] + 1; Q.push(v).
5:     while not Q.empty() do
6:         v ← Q.pop().
7:         if β⃗_s[v] = d_s then continue.
8:         for each (v, c) ∈ E do
9:             if β⃗_s[c] > β⃗_s[v] + 1 then
10:                β⃗_s[c] ← β⃗_s[v] + 1; Q.push(c).
```

(much more complex for deletion)

Experiments



Summary on approximation algorithms for betweenness

- Sampling Rules Everything Around Me;
- Work on pushing down the amount of needed sampling is important;
- Progressive sampling frees us from many worries, but it is challenging;
- Fast and memory efficient data structures are needed to be able to update the estimations fast in dynamic graphs, where approximation is most useful;
- Developing hybrid estimators?

Conclusions

What we presented

- Brief survey of the most common measures of centrality
- Axioms for centrality
- Focusing on closeness and betweenness centrality:
  - exact algorithms on static graphs (GPU-based)
  - exact algorithms on dynamic graphs (streaming, distributed)
  - approximation algorithms for static graphs
  - approximation algorithms for dynamic graphs

In each of the above, there are important open questions and directions for future work.

Big Graphs

- "Big Data" is a lot of hype and refers to very different things depending on the context.
- However, the unprecedented volume, velocity, and variety pose real algorithmic challenges, especially when dealing with expressive and complex representations such as graphs.
- Challenges are opportunities for researchers!
- Big graphs require new algorithms

Volume requires new algorithms

- Classic computational complexity:
  - Is there a polynomial time exact algorithm $\rightarrow$? Go for it!
  - Your problem is **NP**-Hard $\rightarrow$ better think about approximation algorithms...
- Classic computational complexity: polynomial = feasible
- But is polynomial time really feasible?
  - E.g., Brandes algorithm not feasible for $n = 10^9$
- On big graphs quadratic time is as bad as **NP**-Hard
  - New, finer-grain, complexity theory needed (?)
- Need for massively parallel algorithms, out-of-core algorithms, sublinear algorithms, approximated algorithms, randomized algorithms, etc.

Velocity requires new algorithms

- The velocity with which new data keeps arriving. . .
- . . . and the velocity with which the information of interest keeps changing.

- In the case of graphs new edges are formed and old edges might disappear at very high speed.
  - How to maintain the centrality score of all vertices continuously updated?

- Velocity requires streaming algorithms that only read each data point once (or a few time), specialized small-space data structures (sketches) that maintain basic statistics and can be updated on-the-fly, algorithms which are robust to changes in the data, etc.

Variety requires new algorithms

- Variety refers to the richness of different information types to be mixed in the analysis.
- Examples in graphs:
  - Vertices have attributes;
  - Vertices are spatio-temporally localized and keeps moving;
  - Edges have types (colors);
  - Edges have multiple types (a.k.a. multigraphs, multiplex networks, multidimensional networks, etc.);
  - Each edge has associated a time series representing the amount of communication (or activity) along the edge per time unit;
  - ...
- Semantic richness in the data implies complexity in the knowledge we can extract.
- Applications involving "multi-structured" data require the definition of new, ad-hoc, model and patterns ...
- ... and of course, the algorithms to extract them,
- and these new algorithms need to be able to deal with the volume and the velocity!

Big Graphs

- The computational complexity of most existing graph algorithms makes them impractical in today's networks, which are:
  - massive,
  - information-rich, and
  - dynamic.
- In order to scale graph analysis to real-world applications and to keep up with their highly dynamic nature, we need to devise new approaches specifically tailored for modern parallel stream processing engines that run on clusters of shared-nothing commodity hardware.

Thank you!

Francesco Bonchi
http://francescobonchi.com
@FrancescoBonchi

Gianmarco De Francisci Morales
http://gdfm.me
@gdfm7

Matteo Riondato
http://matteo.rionda.to
@teorionda

Slides available at

http://matteo.rionda.to/centrtutorial/